# NEUROSYMBOLIC AUTOMATED STORY GENERATION

A Dissertation
Presented to
The Academic Faculty

by

Lara Jean Martin

In Partial Fulfillment
of the Requirements for the Degree
Human-Centered Computing in the
School of Interactive Computing

Georgia Institute of Technology
May 2021

# NEUROSYMBOLIC AUTOMATED STORY GENERATION

Approved by:


Dr. Mark O. Riedl, Advisor
School of Interactive Computing
*Georgia Institute of Technology*

Dr. Ashok Goel
School of Interactive Computing
*Georgia Institute of Technology*

Dr. Ayanna Howard
School of Interactive Computing
*Georgia Institute of Technology*

Dr. Alan W Black
Language Technologies Institute
*Carnegie Mellon University*

Dr. Devi Parikh
School of Interactive Computing
*Georgia Institute of Technology*


Date Approved: December 7, 2020

To everyone who believed in me

# ACKNOWLEDGEMENTS

Thank you to all the various mentors I have had through the years, whether formal or informal. Your help and guidance have been an invaluable asset to me throughout my graduate career. Thank you to my committee members who have helped shape my thesis into the beauty it is today (and who allowed me to join the doctorate club). Thank you to those who noticed and fostered my ability to do research before I started my PhD, particularly my past advisors Shigeto Kawahara, Matthew Stone, and Alan Black.

Thank you to all the friends from my cohort—Chris Purdy, Upol Ehsan (+ Rachel Urban), Ian Stewart, Ceara Byrne, Sid Banerjee, Kelsey Kurzeja, Nolan Wagener, and honorary cohort member Caitlyn Seim—who have kept me (slightly more—or less?) sane these years. I was thankful to find a group of fellow PhD students who were willing to talk about anything else but work for an hour or two. Our PhDinners, boardgame nights, and other random activities were a necessary respite to the daily PhD grind, and in the process, I came out with friends (and colleagues) for life.

Thank you to all the members of the Entertainment Intelligence Lab who have mentored me or have put up with my mentoring over the years. The lab's culture, fostered by the members within it, showed me what an academic community should look like—starting with Matthew Guzdial, Kristen Siu, and Brent Harrison greeting me into the lab and continuing with those who joined after me—Raj Ammanabrolu, Zhiyu Lin, Spencer Fraizer, Sarah Wiegreffe, Jon Balloch, Becky Peng, Amal Alabdulkarim, and all the others from various degree programs who have come to and gone from the EI Lab over the years.

Finally, I would like to give special thanks to: my advisor Mark Riedl, who reminds me that good people can succeed in academia; my parents, Joe & Jeannie Martin, the ones who were always by my side through my ups and downs (I beat the Martin PhD curse!); and my spouse, David Kent, who I would have never even met if I hadn't started this wild journey—and maybe that in itself has made this all worth it. Thank you, all.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| ASTER | Automated Story-Telling with Event Representations |
| ASTER-X | Automated Story-Telling with Event Representations (Causal) eXtension |
| ASTER-XT | Automated Story-Telling with Event Representations (Causal) eXtension with Transformers |
| ⟨ *s, v, o, m* ⟩ | The original event representation: ⟨ *subject, verb, direct object, indirect object or object of a prepositional phrase (modifier)* ⟩ |
| ⟨ *s, v, o, p, m* ⟩ | The updated event representation: ⟨ *subject, verb, direct object, preposition, indirect object or object of a prepositional phrase (modifier)* ⟩ |

# SUMMARY

Although we are currently riding a technological wave of personal assistants, many of these agents still struggle to communicate appropriately. Humans are natural storytellers, so it would be fitting if artificial intelligence (AI) could tell stories as well. Automated story generation is an area of AI research that aims to create agents that tell good stories. With *goodness* being subjective and hard-to-define, I focus on the perceived *coherence* of stories in this thesis. Previous story generation systems use planning and symbolic representations to create new stories, but these systems require a vast amount of knowledge engineering. The stories created by these systems are coherent, but only a finite set of stories can be generated. In contrast, very large neural language models have recently made the headlines in the natural language processing community. Though impressive on the surface, even the most sophisticated of these models begins to lose coherence over time. My research looks at both neural and symbolic techniques of automated story generation. In this dissertation, I created automated story generation systems that improved coherence by leveraging various symbolic approaches for neural systems. I did this through a collection of techniques; by separating out semantic event generation from syntactic sentence generation, manipulating neural event generation to become goal-driven, improving syntactic sentence generation to be more interesting and coherent, and creating a rule-based infrastructure to aid neural networks in causal reasoning.

# CHAPTER 1. WHY STORYTELLING?

Storytelling has been of interest to artificial intelligence (AI) researchers since the earliest days of the field. Artificial intelligence research has addressed story understanding, automated story generation, and the creation of real-time interactive narrative experiences. Specifically, *automated story generation* is the problem of creating a sequence of main plot points for a story in a given domain and with a set of specifications. The domain and specifications can be given by a user or be programmed into the system, and they guide the system toward the type of story it should create.

Beyond entertainment, automated storytelling unlocks the potential for a number of "serious" applications. Computational storytelling systems could engage with forensic investigators, intelligence analysts, or military strategists to hypothesize about crimes or engage in creative war gaming activities. Virtual agents and conversational chatbots can also create a greater sense of rapport with human users by engaging in playful activities or gossip.

The majority of prior work on automated story generation has focused on *closed-world domains*—a virtual world, game, or simulation environment constrained by the set of characters, objects, places, and the actions that can be legally performed in a given state. Such a world can be modeled by finite AI representations, often based on logical formalizations. Because of the amount of modeling by hand that is needed to create these

systems, they are limited to a single topic or would need to be re-represented in order to switch topics.

In contrast, this thesis is about the creation of artificial agents capable of storytelling in open worlds. *Open-world storytelling* is the generation of stories that can theoretically tell a story about any domain—with the action space being anything that can expressed through natural language. The methods for open-world story generation, like much of artificial intelligence, can generally be divided into two categories: *symbolic*/discrete or *neural*/probabilistic. In the next chapter (CHAPTER 2), I will give an overview of what past research has been done on the task of automated story generation, from both symbolic and neural systems.

Despite its potential use in our lives, artificial intelligence research has only begun to explore neural automated story generation, let alone express its limitations. We have seen that neural language models can give us a big advantage when it comes fluent text generation [1], but there are also significant flaws in their ability to generate—especially when generating stories [2]. In this thesis, I will describe my work in CHAPTER 3 through CHAPTER 6 for improving neural story generation by using symbolic augmentations. This brings me to my thesis statement, which says that:

**The perceived coherence of stories produced by neural-based automated story generation systems can be improved by incorporating symbolic approaches—such as schemas, goals, and causal reasoning.**

*Coherence* is an ill-defined term in the field of natural language generation (NLG). In fact, most research does not define it at all, and instead, only describe the methods they used to measure it [3], [4]. But even these methods are inconsistent within the NLG community as some [5], [6] point out. None of this is the fault of any particular research group but an issue with the field still trying to converge on a single definition for a complex idea. In the next couple of paragraphs, I offer a definition of coherence to the NLG community by touching briefly on discourse analysis theory & psychology and pulling together the reasoning behind existing coherence metrics in NLG.

First, I would like to start with what I mean by perceived coherence. If we turn toward discourse analysis, Jerry Hobbs stated that *coherence* "can be partially characterized by a set of coherence relations [within a text], motivated ultimately by the speaker's or writer's need to be understood" [7]. Since an AI has no need to be understood (or any needs at all), I instead look at *perceived coherence*, putting the onus on the reader to decide if they are coming to an understanding of the text. Bear in mind that the designer of the natural language generator could have some idea of what type of text they want the system to generate, but the final text is ultimately generated by the AI.

Discourse analysis also provides us with discourse structure theory [8]. In this theory, it says that a discourse is divided into *segments* with relations between them. This—paired with centering theory—created a hierarchical structure in which coherence can be found both locally (within a segment) and globally (across segments) [9]. Work in NLG also has

gravitated toward measuring local and global coherence—which I will called local and global consistency to prevent a circular definition of coherence—by measuring the *semantic relatedness* between sentences (local consistency) [10]–[13] and *sustaining meaning* & topic modeling (global consistency) [14]–[16].

But what is *semantic relatedness*? When the field of psychology had investigated narrative comprehension [17]–[19], they pointed to the importance of *causal relations*—when human readers model the relationships between story events—for coherence. In narrative, a causal relation between two events is an indication that the temporally latter event is made possible in part by the presence of the temporally former event. This can be due to direct cause—a wheel falling off a car causes it to crash—or through enablement—the car could not have crashed if the ignition had not been turned on. Symbolic story planning systems directly model enablement [20], [21] and sometimes also the direct cause of character intentions [22], at the expense of being closed-world generators.

The *narrative causality* just described differs from the causality as used in statistics or Bayesian inference, where causality between random variables means that the value of one random variable affects the distribution of values from another random variable. This *Bayesian causality* limits neural story generators to solely temporal orderings without a guarantee of narrative causal relations.

Once we know what sentences belong together, they then need to be in a particular order. Xu et al. call coherence "a discourse property that is concerned with the *logical and*

*semantic organization* [emphasis added] of a passage, such that the overall meaning of the passage is expressed fluidly and clearly" [23]. This can be interpreted as the *ordering* of sentences (causal ordering) [24]–[27].

**This brings me to my definition of *coherence*: it is the measure of the local and global consistency of textual segments along with the causal/logical ordering of sentences.** In other words, *a story* is coherent if it contains high levels of local and global consistency, and its sentences are ordered properly. There are and will be other ways that people will define the dimensions of coherence. (Porzel & Gurevych [28] use ontological, situational, and contextual coherence.) However, I found the three aspects I list in my definition to be fairly widespread and used consistently throughout the NLP community.

I begin my dissertation by presenting my work on improving plot generation (CHAPTER 3). I did this by extracting an event schema to represent the meaning of a sentence, enabling me to generate plot points separate from the syntax of the sentences—which improves the *local consistency* of the story. This created two new problems: how can I use these events to create a goal-driven narrative (CHAPTER 4), and how can I translate these events back into readable English sentences (CHAPTER 5)? I solved the former by retraining a neural network using a technique inspired by reinforcement learning to guide it through a progression of events, improving coherence through global consistency. The latter I solved by creating a cascading ensemble of methods that balanced maintaining the meaning of the event while also having interesting language (i.e.,

maintaining local consistency while being enjoyable). Despite all this, I saw that the stories generated by these systems could still be more coherent, particularly in terms of the logical ordering of events. By leveraging the flexibility of a neural network model with rule-based constraints, I have created a system that is capable of telling stories that are not just temporally ordered but causally ordered (CHAPTER 6). These four chapters and their relation to coherence are summarized in Table 1. Through this work I have shown that by creating storytelling agents via neurosymbolic methods, they become capable of creating more locally- & globally-consistent and logically-ordered stories than a neural network is able to do alone.

**Table 1 – An outline of the types of neurosymbolic story generation techniques presented in this dissertation, along with the primary aspect of perceived coherence that was improved by each technique.**

| *Chapter* | *Topic* | *Aspect of Perceived Coherence* |
|---|---|---|
| *3* | Events/Schemata | Improving local consistency |
| *4* | Goal-driven generation | Improving global consistency |
| *5* | Event-to-sentence translation | Maintaining local consistency |
| *6* | Causal generation | Improving logical ordering |

# CHAPTER 2. ONCE UPON A TIME: A BRIEF HISTORY OF AUTOMATED STORY GENERATION

Humans have the ability to connect seemingly unrelated ideas together. If a computer is working together with a user to create a new story, the AI must be prepared to handle anything the human can think of. Yet computers do not easily lend themselves to storytelling in such an open space. As described in the first chapter, *automated story generation* has been a research problem of interest since nearly the inception of artificial intelligence. It is the problem of automatically selecting a sequence of events, actions, or words that can be told as a story. In this chapter, I will discuss previous closed-world & open-world symbolic story generation and neural (open-world) story generation efforts.

## 2.1 Interactive Narrative & Closed Worlds

Previously, most story generation systems have used symbolic planning [21] or case-based reasoning [29]. While these automated story generation systems were able to produce impressive results, they rely on a human-knowledge engineer to provide symbolic domain models that indicated legal characters, actions, and knowledge about when character actions can and cannot be performed. These techniques could only generate stories for predetermined and well-defined domains. However, the creativity of these systems conflated the robustness of manually-engineered knowledge and algorithm suitability.

Previous causal-based automated storytelling systems required a lot of hand holding. Riedl and Bulitko [30] give an overview of AI approaches to interactive narrative. The most common form of interactive narrative involves the user taking on the role of the protagonist in an unfolding storyline. The user can also be a disembodied observer—as if watching a movie—but capable of making changes to the world or talking to the characters [31]. A common solution, first proposed by Bates [32] is to implement a *drama manager*. A drama manager is an intelligent, omniscient, and disembodied agent that monitors the virtual world and intervenes to drive the narrative forward according to some model of quality of experience. An *experience manager* [33] is a generalization of this concept, recognizing the fact that not all narratives need to be dramatic, such as in the case of education or training applications.

To appropriately manage narratives, experience managers need to be story generators. There are many AI approaches to the problem of story generation. One way is to treat it as a form of search such as planning [31], [33]–[37], adversarial search [38], [39], reinforcement learning [40], or case-based reasoning [41], [42]. All of the above systems assume an *a priori*-known domain model that defines what actions are available to a character at any given time. Some of the earliest systems like TALE-SPIN (1977) [20] or UNIVERSE (1984) [43] would plan toward narrative goals. TALE-SPIN—and many of the planning story generation systems since—focused on the goals of *consistency* and *coherence*. One way to do this is through *causality*. By adding causal links between plot points in a story, these systems were able to generate stories that were consistent to the

facts of the real world. *Causal links* are connections specifying what conditions need to be satisfied in order to continue on to the connecting plot point (e.g., the knight needs to equip her sword before slaying the dragon). They are the tools used to address narrative causality in some planning systems, and they can take many forms but are often distilled into effects & requirements.

Closed-world systems have been created that appear open. Façade [44] allows users to interact with virtual characters by freely inputting text. This gives the *appearance* of open communication between the human player and the virtual world; however, the system limits interactions by assigning natural language sentences to dramatic beats that are part of the domain model. Farrell, Ware, & Baker [36] use Indexter [45] to calculate salience of possible events to determine which would be the most appropriate to display to the user. This cuts down on the amount of connections they need to develop in their planning model, but the events are still hand-authored and fixed to a domain.

## 2.2 Open World Generation

*Open story generation* [46] is the problem of automatically generating a story about any domain without a priori manual knowledge engineering. Open story generation requires an intelligent system to either learn a domain model from available data [46], [47] or to reuse data and knowledge available from a corpus, whether already decomposed into symbolic facts [48] or not [49].

Open-world story generation attempts to break the assumption of an *a priori*-known domain model. The *Scheherazade* system [46] uses a crowdsourced corpus of example stories to learn a domain model from which to generate novel stories. Scheherazade-IF [50] attempts to learn a domain model in order to create new stories and interactive experiences in previously unknown domains. However, once the domain is learned, it is limited in what actions the human can perform to those in the domain model. Say Anything [49] is a textual case-based-reasoning story generator, meaning that it operates in the space of possible natural language sentences. It responds to the human user by finding sentences in blogs that share a similarity to human-provided sentences, but consequently tends to fail to maintain story coherence. Open-world story generation can also be done probabilistically using machine learning.

## 2.3 Deep Learning for Story Generation

Deep learning approaches to automated plot generation can learn storytelling and domain knowledge from a corpus of existing stories or plot summaries, from which stories can be created or segments of story content can be identified in existing repositories to be assembled into stories.

Some of the earliest neural story generators used recurrent neural networks (RNNs), which can theoretically learn to predict the probability of the next character, word, or sentence in a story. Roemmele and Gordon [47] used a Long Short-Term Memory (LSTM) network [51] to generate stories. They used Skip-thought vectors [52] to encode sentences

and a technique similar to word2vec [53] to embedded entire sentences into 4,800-dimensional space. However, while RNNs using LSTM or gated recurrent unit (GRU) cells can theoretically maintain long-term context in their hidden layers, in practice RNNs only use a relatively small part of the history of tokens [54]. Consequently, stories or plots generated by RNNs tend to lose coherence as the generation continues. Khalifa et al. [55] argue that stories are better generated using recurrent neural networks trained on highly specialized textual corpora, such as the body of works from a single, prolific author. However, this limits the scope of the types of stories that the system can generate.

Even as we are able to leverage the power of our computers' hardware more and train larger models, these models still lose coherence. Very large language models, such as GPT-2 (Generative Pre-trained Transformer 2), have been shown to work well with a variety of short-term tasks like understanding short children's stories [1] but run into this coherence pitfall if the stories get much longer.

Neural language modeling-based story generation approaches [55]–[59] are also prone to generating stories without long-term direction. Since each sentence, event, word, or letter is generated by sampling from a probability distribution, these models by themselves lack any goal. Inspired in-part by the work I will present in CHAPTER 3, Fan et al. [60] created hierarchical neural networks to first generate the story *prompt* and then generating the full story from it. Others [4], [61], [62] have also decomposed story generation into levels of different abstraction by presenting a storyline and generating a story from it,

including work by Shen et al. [63] who used transformers with the event representation I will describe in the next chapter.

Related to automated story generation, the *story cloze test* [64] is the task of choosing between two given endings to a story. The story cloze test transforms story generation into a classification problem: a 4-sentence story is given along with two alternative sentences that can be the 5th sentence. State-of-the art story cloze test techniques use a combination of word embeddings, sentiment analysis, and stylistic features [65].

There has also been work with *interactive* neural story generation. These systems are designed to use neural networks to generate text that humans will interact with and manipulate in real time. Like Roemmele & Gordon [66] did with case-based reasoning, Clark et al. [67] created a writing helper, a network that would provide text that can be kept, altered, or discarded completely. Users would be encouraged to ask the system for the next sentence of the story so that it would inspire their writing. Interactive story generation systems can easily be used in games as well. Like a more sophisticated AI Dungeon[1], Ammanabrolu et al. [68] developed a system that creates interactive fiction

[1] https://play.aidungeon.io

games using dynamic knowledge graphs from a system they dub AskBERT, a Question-Answering version of the model ALBERT [69].

In this thesis, I begin with the use of recurrent encoder-decoder neural networks (e.g., Sequence-to-Sequence [70]) for open story generation. Very large language models have shown to generate stories with decent grammar [1]. However, even with so much data these models will still lose context within a short story. Therefore, I begin by separating the problems of grammar generation from story event generation.

# CHAPTER 3. SEPARATING SEMANTICS FROM SYNTAX

Neural language models, such as encoder-decoder RNNs, are trained to predict the next token(s) in a sequence, given one or more input tokens. The network architecture and set of weights $\theta$ comprise a generative model capturing and generalizing over patterns observed in a training corpus. For this work, I used a corpus of movie plot summaries extracted from Wikipedia [71] under the premise that the set of movie plots on Wikipedia covers the range of domains that people want to tell stories about. The goal being to create a story that reads like a movie plot, capturing both the style and the plot progression. However, neural networks have a tendency to learn grammar and style (word usage) before being able to generate reasonable story events.

In narratological terms, an *event* is a unit of story featuring a world state change [72]. Textual story corpora, including the Wikipedia movie plot corpus, are comprised of unstructured textual sentences. One benefit to dealing with movie plots is the clarity of events that occur, although this is often at the expense of more creative language. Even so, character- or word-level analysis of these sentences would fail to capture the interplay between the words that make up the meaning behind the sentence. Character- and word-level recurrent neural networks can learn to create grammatically-correct sentences but often fail to produce coherent narratives beyond a couple of sentences (local consistency). On the other hand, sentence-level events would be too unique from each other to find any real relationship between them. Even with a large corpus of stories, we would most likely

have sequences of sentences that would only ever be seen once. For example, “Old ranch-hand Frank Osorio travels from Patagonia to Buenos Aires to bring the news of his daughter’s demise to his granddaughter Alina.” occurs only once in the entire corpus of movie plots, so we have only ever seen one example of what is likely to occur before and after it (if anything). Due to event sparsity, my model would be likely to have poor predictive ability. In order to help maintain a coherent story, one can provide an event representation that is expressive enough to preserve the semantic meaning of sentences in a story corpus while also reducing the sparsity of events. The reduction in sparsity ends up increasing the potential overlap of events across stories and the number of examples of events the learner observes. Having the events seen more often throughout the corpus aids in the neural network’s ability to learn patterns of events, potentially improving the coherence of the generated text.

In this chapter, I will discuss the development of an event representation that aids in the process of automated, open story generation. The insight is that if one can extract some basic semantic information from the sentences of preexisting stories, one can learn the skeletons of what “good” stories are supposed to be like. Then, using these templates, the system can generate novel sequences of events that would resemble a decent story. In addition to the event representation, I also propose in this chapter a recurrent encoder-decoder neural network for story generation called *event-to-event*. I evaluated my event representation against the naive baseline sentence representation and a number of alternative representations. I measured the ability to predict the true successor event as an

indicator of how event representations facilitate the modeling of context. In *event-to-event*, a textual story corpus is preprocessed—sentences are translated into my event representation by extracting the core semantic information from each sentence. Event preprocessing is a linear-time algorithm using a number of natural language processing techniques. The processed text is then used to train the neural network. However, event preprocessing is a lossy process, and the resultant events are not human-readable. To address this, I present a story generation pipeline in which a second neural network, *event-to-sentence*, translates abstract events back into natural language sentences. The *event-to-sentence* network is an encoder-decoder network trained to fill in the missing details necessary for the abstract events to be human-readable. This chapter also introduces an overall story generation pipeline in which subsequent events of a story are generated via an *event-to-event* network and then translated into natural language using an *event-to-sentence* network. I present an evaluation of *event-to-sentence* on different event representations and draw conclusions about the effect of representations on the ability to produce readable stories.

## 3.1 Events

*Automated story generation* can be formalized as follows: given a sequence of events, sample from the probability distribution over successor events. That is, simple automated story generation can be expressed as a process whereby the next event is computed by sampling from or maximizing $P_\theta(e_{t+1}|e_{t-k}, ..., e_{t-1}, e_t)$ where $\theta$ is the set of parameters of a

generative domain model, $e_i$ is the event at time $i$, and $k$ indicates the size of a sliding window of context, or history. In my work, the probability distribution is produced by a recurrent encoder-decoder network with parameters $\theta$. In this section, I consider what the level of abstraction for the inputs into the network should be such that it produces the best predictive power while retaining semantic knowledge.

Based on the theory of script learning [73], Chambers and Jurafsky [74] learn causal chains that revolve around a protagonist. They developed a representation that took note of the event/verb and the type of dependency that connected the event to the protagonist (e.g. was the protagonist the object of this event?). Also for scripts, Pichotta and Mooney [75] developed a 5-tuple event representation of $\langle v, e_s, e_o, e_{ps}, p \rangle$, where $v$ is the verb, $p$ is a preposition, and $e_s$, $e_o$, and $e_p$ are nouns representing the subject, direction object, and prepositional object, respectively. Because it was a paper on script learning, they did not need to convert the event representations back into natural language. My representation is inspired by this work.

For the work in this chapter, I use a corpus of movie plot summaries extracted from Wikipedia by Bamman et al. [71], cleaned to remove any extraneous Wikipedia syntax, such as links for which actors played which characters. This corpus is used under the premise that the set of movie plots on Wikipedia covers the range of domains that people want to tell stories about. The corpus contains 42,170 stories with an average number of 14.515 sentences per story.

## 3.2 A Representation for Events

Event sparsity results in a situation where all event successors have a low probability of occurrence, potentially within a margin of error. In this situation, story generation devolves to a random generation process. Following Pichotta and Mooney [75], I developed a 4-tuple event representation[2]

$$\langle s, v, o, m \rangle$$

where $v$ is a verb, $s$ is the subject of the verb, $o$ is the object of the verb, and $m$ is the modifier—or "wildcard", which can be a propositional object, indirect object, causal complement (e.g., in "I was glad that he drove," "drove" is the causal complement to "glad."), or any other dependency unclassifiable to Stanford's dependency parser. An example of an eventified sentence is shown in Figure 1. All words were stemmed. Events were created by first extracting dependencies with Stanford's CoreNLP [76] and locating the appropriate dependencies mentioned above. If the object or modifier cannot be identified, I insert the placeholder *EmptyParameter*, which I will refer to as $\emptyset$. My event translation process can either extract a single event from a sentence or multiple events per

[2] This 4-tuple representation is later (in CHAPTER 5 & CHAPTER 6) expanded to include the preposition $p$ for the prepositional phrase: $\langle s, v, o, p, m \rangle$.

sentence. If I were to extract multiple events, it is because there are verbal or sentential conjunctions in the sentence. Consider the sentence "John and Mary went to the store," my algorithm would extract two events: ⟨*john, go, store, ∅*⟩ and ⟨*mary, go, store, ∅*⟩. The average number of events per sentence was 2.69. The simplest form of my event representation is achieved by extracting the verb, subject, object, and modifier term from each sentence.

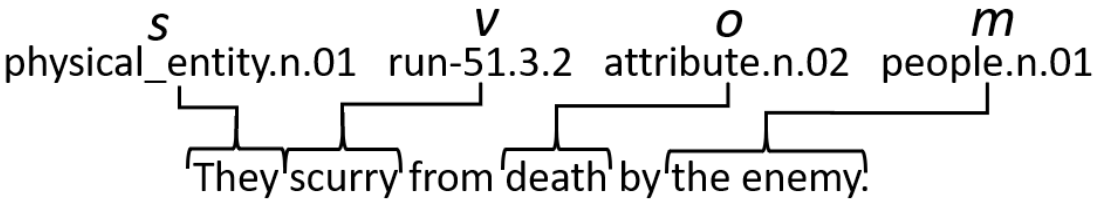


**Figure 1 – An example sentence and its representation as a generalized event.**

However, there are variations on the event representation that increase the level of abstraction (and thus decrease sparsity) and help the encoder-decoder network predict successor events. I enumerate some of the possible variations below.

- *Generalized.* Each element in the event tuple undergoes further abstraction. Named entities were identified (cf. [77]), and "PERSON" names were replaced with the tag <*PERSON*>*n*, where *n* indicates the $n^{th}$ character name in the sentence. Other named entities were labelled as their named entity recognition (NER) category (e.g. LOCATION, ORGANIZATION, etc.). The rest of the nouns were replaced by the WordNet [78] *Synset* two levels up in the inherited hypernym hierarchy, giving us

a general category (e.g. self-propelled vehicle.n.01 vs the original word "car" (car.n.01)), while avoiding labelling it too generally (e.g. entity.n.01). Verbs were replaced by VerbNet [79] version 3.2.4[3] frames (e.g. "arrived"/"arriving" become "escape-51.1").

- *Character Name Numbering.* There were two ways of numbering the character names that I experimented with. One way had the character name numbering reset with every sentence (consistent within sentence)–or, *sentence* PERSONs, my "default". The other way had the numbering reset after every input-output pair (i.e. consistent across two sentences)–or, *continued* PERSONs.
- *Adding Genre Information.* I ran topic-modelling on the entire corpus using Python's Latent Dirichlet Analysis[4] set for discovering 100 different categories. I took this categorization as a type of emergent genre classification. Some clusters had a clear pattern, e.g., "job company work money business". Others were less clear. Each cluster was given a unique genre number which was added to the event representation to create a 5-tuple $\langle s, v, o, m, g \rangle$ where *s, v, o,* and *m* are defined as above and *g* is the genre cluster number.

[3] https://verbs.colorado.edu/vn3.2.4-test-uvi/index.php
[4] https://pypi.python.org/pypi/lda

Note that other event representations can exist, including representations that incorporate more information as in Pichotta & Mooney [75]. The experiments in the next section show how different representations affect the ability of a recurrent neural network to predict story continuations.

### 3.3 Event-to-Event

The event-to-event network is a recurrent multi-layer encoder-decoder network based on [70]. Unless otherwise stated in experiments below, my event-to-event network is trained with input $x = w_1^n, w_2^n, w_3^n, w_4^n$ where each $w_i^n$ is either *s*, *v*, *o*, or *m* from the *n*-th event, and output $y = w_1^{n+1}, w_2^{n+1}, w_3^{n+1}, w_4^{n+1}$. The experiments described below seek to determine how different event representations affected event-to-event predictions of the successor event in a story. I evaluated each event representation using two metrics. Perplexity is the measure of how "surprised" a model is by a training set. Here I use it to gain a sense of how well the probabilistic model I have trained can predict the data. Specifically, I built the model using an *n*-gram length of 1:

$$Perplexity = 2^{-\sum_x p(x) \log_2 p(x)} \quad (1)$$

where *x* is a token in the text, and

$$p(x) = \frac{count(x)}{\sum_{y \epsilon Y} count(y)} \quad (2)$$

where *Y* is the vocabulary. The larger the unigram perplexity, the less likely a model is to produce the next unigram in a test dataset. The second metric is BLEU score, which compares the similarity between the generated output and the "ground truth" by looking at n-gram precision. The neural network architecture I used was initially envisioned for machine translation purposes, where BLEU is a common evaluation metric. Specifically, I use an *n*-gram length of 4 and so the score takes into account all *n*-gram overlaps between the generated and expected output where *n* varies from 1 to 4 [80]. I use a greedy decoder to produce the final sequence by taking the token with the highest probability at each step.

$$\widehat{W} = arg \max_{w} P(w|S) \tag{3}$$

where $\widehat{W}$ is the generated token appended to the hypothesis, *S* is the input sequence, and w represents the possible output tokens.

### 3.4 Event-to-Event Evaluation

For each experiment, I trained a sequence-to-sequence recurrent neural net [70] using Tensorflow [81]. Each network was trained with the same parameters (0.5 learning rate, 0.99 learning rate decay, 5.0 maximum gradient, 64 batch size, 1024 model layer size, and 4 layers), varying only the input/output, the bucket size, the number of epochs and the vocabulary. The neural nets were trained until the decrease in overall loss was less than 5%

per epoch. This took between 40 to 60 epochs for all experiments. The data was split into 80% training, 10% validation, and 10% test data. All reported results were evaluated using the held-out test data. I evaluated 11 versions of my event representation against a sentence-level baseline. Numbers below correspond to rows in results Table 2.

0. *Original Sentences.* As my baseline, I evaluated how well an original sentence can predict its following original sentence within a story.
1. *Original Words Baseline.* I took the most basic, 4-word event representation: ⟨*s, v, o, m*⟩ with no abstraction and using original character names.
2. *Original Words with <PERSON>s.* This experiment is identical to the previous except entity names that were classified as "PERSON" through NER were substituted with <PERSON>n.
3. *Generalized.* Using the same 4-word event structure, I replaced character names and generalized all other words through WordNet or VerbNet, following the procedure described earlier.

To avoid an overwhelming number of experiments, the next set of experiments used the "winner" of the first set of experiments. Subsequent experiments used variations of the generalized event representation (#3), which showed drastically lower perplexity scores.

4. *Generalized, Continued <PERSON>s.* This experiment mirrors the previous with the exception of the number of the <PERSON>s. In the previous experiment, the numbers restarted after every event. Here, the numbers

continue across input and output. For example, if $event_1$ mentioned “Kendall” and $event_2$ (which follows $event_1$ in the story) mentioned “Kendall”, then both would have the same number for this character.

5. *Generalized + Genre.* This is the same event structure as experiment #3 with the exception of an additional, 5th parameter in the event: genre. The genre number was used in training for event-to-event but removed from inputs and outputs before testing; it artificially inflated BLEU scores since it was easy for the network to guess the genre number as the genre number was weighted equally to other words.
6. *Generalized Bigram.* This experiment tests whether RNN history aids in predicting the next event. I modified event-to-event to give it the event bigram $e_{n-1}$, $e_n$ and to predict $e_{n+1}$, $e_{n+2}$. I believe that this experiment could generalize to cases with a history $e_{n-k}$, ..., $e_n$.
7. *Generalized Bigram, Continued <PERSON>s.* This experiment has the same continued <PERSON> numbering as experiment #4 had but with event-to-event trained on event bigrams.
8. *Generalized Bigram + Genre.* This is a combination of the ideas from experiments #5 and #6: generalized events in event bigrams and with genre added.

The following three experiments investigate extracting more than one event per sentence in the story corpus when possible; the prior experiments only use the first event per sentence in the original corpus.

9. *Generalized Multiple, Sequential.* When a sentence yields more than one event, $e_n^1$, $e_n^2$, ... where *n* is the $n^{th}$ and $e_n^i$ is the $i^{th}$ event created from the $n^{th}$ sentence, I train the neural network as if each event occurs in sequence, i.e., $e_n^1$ predicts $e_n^2$, $e_n^2$ predicts $e_n^3$, etc. The last event from sentence *n* predicts the first event from sentence *n* + 1.
10. *Generalized Multiple, Any Order.* Here I gave the RNN all orderings of the events produced by a single sentence paired, in turn, with all orderings of each event of the following sentence.
11. *Generalized Multiple, All to All.* In this experiment, I took all of the events produced by a single sentence together as the input, with all of the events produced by its following sentence together as output. For example, if sentence *i* produced events $e_i^1, e_i^2$, and $e_i^3$, and the following sentence *j* produced events $e_j^1$ and $e_j^2$, then I would train the neural network on the input: $e_i^1, e_i^2, e_i^3$ and the output: $e_j^1, e_j^2$.

**Table 2 – Results from the event-to-event experiments. Best values from each of these three sections (baselines, additions, and multiple events) are bolded.**

| Experiment | Perplexity | BLEU |
|---|---|---|
| (0) Original Sentences | 704.815 | 0.0432 |
| (1) Original Words Baseline | 748.914 | **0.1880** |
| (2) Original Words with PERSONs | 166.646 | 0.1878 |
| (3) Generalized Baseline | **54.231** | 0.0575 |
| (4) Generalized, Continued PERSONs | 56.180 | 0.0544 |
| (5) Generalized + Genre | **48.041** | 0.0525 |
| (6) Generalized Bigram | 50.636 | 0.1549 |
| (7) Generalized Bigram, Continued PERSONs | 50.189 | **0.1567** |
| (8) Generalized Bigram + Genre | 48.505 | 0.1102 |
| (9) Generalized Multiple, Sequential | 58.562 | 0.0521 |
| (10) Generalized Multiple, Any Order | 61.532 | 0.0405 |
| (11) Generalized Multiple, All to All | **45.223** | **0.1091** |

### *3.4.1 Results and Discussion*

The results from the experiments outlined above can be found in Table 2. The original word events had similar perplexity to original sentences. This parallels similar observations made by Pichotta and Mooney [82]. Deleting words did little to improve the predictive ability of my event-to-event network. However, perplexity improved significantly once character names were replaced by generalized <PERSON> tags, followed by generalizing other words. Overall, the generalized events had much better perplexity scores, and making them into bigrams—incorporating history—improved the BLEU scores to nearly those of the original word events. Adding in genre information improved perplexity. The best perplexity was achieved when multiple generalized events were created from sentences as long as all of the events were fed in at the same time (i.e. no order was being forced upon the events that came from the same sentence). The training data was set up to encourage

the neural network to correlate all of the events in one sentence with all of the events from the next sentence. Although the events with the original words (with or without character names) performed better in terms of BLEU score, it is my belief that BLEU is not the most appropriate metric for event generation because it emphasizes the recreation of the input. It is possible to have a good event continuation without any of the target words.

Overall, BLEU scores are very low for all experiments, attesting to the inappropriateness of the metric. Perplexity is a more appropriate metric for event generation because it correlates with the ability for a model to predict the entire test dataset. Borrowing heavily from the field of language modelling, the recurrent neural network approach to story generation is a prediction problem. My intuition that the generalized events would perform better in generating successive events bears out in the data. However, greater generalization makes it harder to return events to natural language sentences. I also see that the BLEU scores for the bigram experiments are generally higher than the others. This shows that history matters, and that the additional context increases the number of n-gram overlaps between the generated and expected outputs. The movie plots corpus contains numerous sentences that can be interpreted as describing multiple events. Naive implementation of multiple events hurt perplexity because there is no implicit order of events generated from the same sentence; they are not necessarily sequential. When I allow multiple events from sentences to be followed by all of the events from a subsequent sentence, perplexity improves.

### 3.5 Event-to-Sentence

Unfortunately, events are not human-readable and must be converted to natural language sentences. Since the conversion from sentences to (multiple) events for event-to-event is a linear and lossy process, the translation of events back to sentences is non-trivial as it requires adding details back in. For example, the event ⟨ *relative.n.01, characterize-29.2, male.n.02, feeling.n.01* ⟩ could, hypothetically, have come from the sentence "Her brother praised the boy for his empathy." In actuality, this event was found in the corpus extracted from the sentence "His uncle however regards him with disgust."

Complicating the situation, the event-to-event encoder-decoder network is not guaranteed to produce an event that has ever been seen in the training story corpus. Furthermore, my experiments with event representations for event-to-event indicate that greater generalization lends to better story generation. However, the greater the generalization, the harder it is to translate an event back into a natural language sentence. In this section I introduce event-to-sentence, a neural network designed to translate an event into natural language. The event-to-event network takes an input event $e_n$ = ⟨ $s_n$, $v_n$, $o_n$, $v_n$ ⟩ and samples from a distribution over possible successor events $e_{n+1}$ = ⟨ $s_{n+1}$, $v_{n+1}$, $o_{n+1}$, $m_{n+1}$ ⟩. As before, I use a recurrent encoder-decoder network based on [70]. The event-to-sentence network is trained on parallel corpora of sentences from a story corpus and the corresponding events. In that sense, event-to-sentence is attempting to learn to reverse the lossy event creation process.

### 3.6 Event-to-Sentence Evaluation

The setup for this set of experiments is almost identical to that of the event-to-event experiments, with the main difference being that I used PyTorch which more easily lent itself to implementing beam search. The LSTM RNN networks in these experiments use beam search instead of greedy search to aid in finding a more optimal solution while decoding. The beam search decoder works by maintaining a number of partial hypotheses at each step (known as the beam width or B, where B=5). Each of these hypotheses is a potential prefix of a sentence. At each step, the B tokens with the highest probabilities in the distribution are used to expand the partial hypotheses. This continues until the end-of-sentence tag is reached. The input for these experiments was the events of a particular representation and the output was a newly-generated sentence based on the input event(s). The models in these experiments were trained on the events paired with the sentences they were "eventified" from. In a complete story generation system, the output of the event-to-event network feeds into the event-to-sentence network. Examples of this can be seen in Table 4. However, I tested the event-to-sentence network on the same events that were used for the event-to-event experiments in order to conduct controlled experiments–I know the sentences from which they came–and compute perplexity and BLEU.

To test event-to-sentence with an event representation that used the original words is relatively straight forward. Experimenting on translating generalized events to natural language sentences was more challenging since I would be forcing the neural net to guess

character names, other nouns, and verbs. I devised an alternative approach for generalized event-to-sentence whereby sentences were first partially eventified. That is, I trained event-to-sentence on generalized sentences where the “PERSON” named entities were replaced by <PERSON> tags, other named entities were replaced by their NER category, and the remaining nouns were replaced by WordNet *Synsets* up two levels in WordNet’s hierarchy. Verbs were left alone since they often do not have to be consistent across sentences within a story. The intuition here is that the character names and particulars of objects and places are highly mutable and do not affect the overall flow of a story as long as they remain consistent. Below, I show an example of a sentence and its partially generalized counterpart. The original sentence

> *The remaining craft launches a Buzz droid at the ARC 1 7 0 which lands near the Clone Trooper rear gunner who uses a can of Buzz Spray to dislodge the robot.*

would be partially generalized to

> *The remaining activity.n.01 launches a happening.n.01 droid at the ORGANIZATION 1 7 0 which property.n.01 near the person.n.01 enlisted person.n.01 rear skilled worker.n.01 who uses a instrumentality.n.03 of happening.n.01 chemical.n.01 to dislodge the device.n.01*

I also looked at whether event-to-sentence performance would be improved if I used multiple events per sentence (when possible) instead of the default single event per

sentence. Alternatively, I automatically split and prune (S+P) sentences; removing prepositional phrases, splitting sentential phrases on conjunctions, and, when it does not start with a pronoun (e.g. who), splitting S' (read: S-bar) from its original sentence and removing the first word. This would allow me to evaluate sentences that would have fewer (ideally one) events extracted from each. For example,

> *Lenny begins to walk away but Sam insults him so he turns and fires, but the gun explodes in his hand.*

becomes

> *Lenny begins to walk away. Sam insults him. He turns and fires. The gun explodes.*

Although splitting and pruning the sentences should bring most sentences down to a single event, this is not always the case. Thus, I ran an event-to-sentence experiment where I extracted all of the events from the S+P sentences.

### *3.6.1 Results and Discussion*

The results of my event-to-sentence experiments are shown in Table 3. Although generalizing sentences improves perplexity drastically, splitting and pruning sentences yields better BLEU scores when the original words are kept. In the case of event-to-sentence, BLEU scores make more sense as a metric since the task is a translation task. Perplexity in these experiments appears to correspond to vocabulary size. Generalized

events with full-length generalized sentences have better BLEU scores than when the original words are used. However, when I work with S+P sentences, the pattern flips. I believe that because both S+P and word generalizing methods reduce sparsity of events, when they are combined too much information is lost and the neural network struggles to find any distinguishing patterns. Table 4 shows examples from the entire pipeline without slot filling (See Figure 2 for an illustration of the pipeline). To get a full sense of how the generalized sentences would read, imagine adding character names and other details as if one were completing a Mad-Libs game.

**Table 3 – Results from the event-to-sentence experiments.**

| **Experiment** | **Perplexity** | **BLEU** |
|---|---|---|
| Original Words Event → Original Sentence | 1585.46 | 0.0016 |
| Generalized Event → Generalized Sentence | 56.516 | 0.0331 |
| All Generalized Events → Generalized Sentence | 59.106 | 0.0366 |
| Original Words Event → S+P Sentence | 490.010 | **0.0764** |
| Generalized Event → S+P Generalized Sentence | **53.964** | 0.0266 |
| All Generalized Events →S+P Generalized Sentence | 56.488 | 0.0283 |

I envisioned event-to-event and event-to-sentence working together as illustrated in Figure 2. I refer to this pipeline as the **ASTER** system: *Automated Story-Telling with Event Representations*. First, a sentence—provided by a human—is turned into one or more events. The event-to-event network generates one or more successive events. The event-to-sentence network translates the events back into natural language and presents it to the human reader. To continue story generation, $event_{n+1}$ can be fed back into event-to-event; the sentence generation is purely for human consumption. Once sentences are generated,

they need to be filled with specific character names and entities appropriate for the current story. A Slot-Filler, with help from a Memory component, is responsible for adapting the sentence to the specific context. Further experimentation on the *event-to-sentence* component of the pipeline can be found in CHAPTER 5.

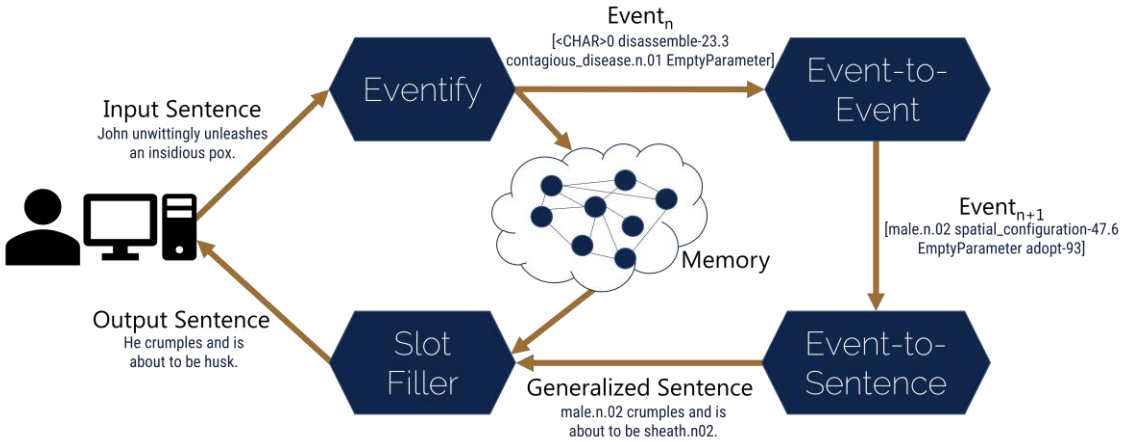


**Figure 2 – The end-to-end pipeline, taking in an input sentence from the user and outputting the next sentence of the story.**

### 3.7 Working & Long-Term Memory

Generalized categories are useful for adapting generated text to a variety of topics, but the question remains how to determine exactly what character names and noun details to use in place for the <*PERSON*>s and WordNet placeholders. I proposed the addition of Working Memory and Long-Term Memory modules. The Working Memory module retains the character names and nouns that were removed during the eventification process. The names and nouns are then retrieved at a later time during the story. After a partially

generalized sentence is produced by event-to-sentence, the system uses the Working Memory lookup table to fill character names and nouns back into the placeholders. The intuition is that from one event to the next, many of the details—especially character names—are likely to be reused. In stories it is common to see a form of turn-taking between characters. For example, the two events "John hits Andrew" & "Andrew runs away from John" followed by "John chases Andrew" illustrate the turn-taking pattern. If John was always used as the first character name, the meaning of the example would be significantly altered. The continuous numbering of character names (Event-to-Event experiment #7; see Section 3.4, pg. 24) is designed to assist event bigrams with maintaining turn-taking patterns. There are times when the Working Memory will not be able to fill character name and WordNet *Synset* placeholder slots because the most recent event bigram does not contain the element necessary for reuse. The Long-Term Memory maintains a history of all character names and nouns that have ever been used in the story and information about how long ago they were last used by storing them in a dynamic graph (see Figure 3).

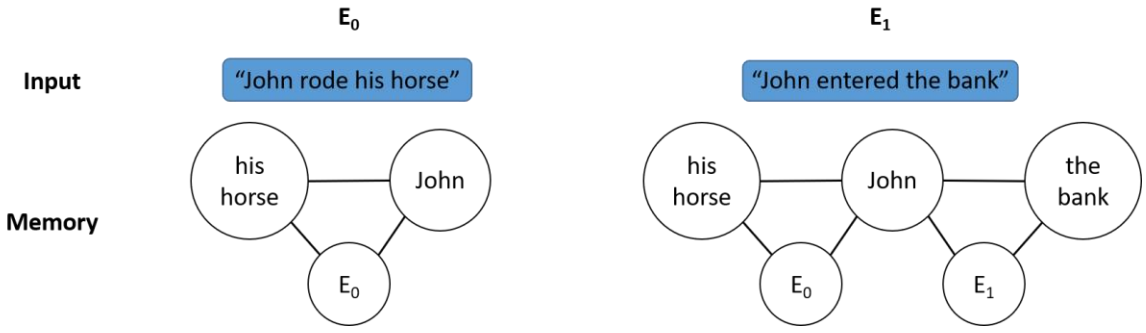


**Figure 3 – An example of the dynamic memory graph over the course of two events.**

The AI's memory allows the agent to organize concepts based on recency. I proposed to use the *event-indexing model* [83], a psychology framework that is meant to explain how people comprehend new information and store it in their long-term memory. Event indexing has been modified previously for artificial intelligence and story comprehension to model a reader's memory [84], [85]. The AI maintains a memory graph of which nouns (entities) were mentioned during which turns. When the objects are mentioned during an action, they are connected to that event's node $E_0$ to $E_n$, where $n$ is the number of the event in the story when the entity is mentioned (Figure 3). The salience of objects can be computed as the distance in the memory graph between two objects. The underlying assumption is that stories are more likely to reuse existing entities and concepts than introduce new ones.

### 3.8 Slot-Filling

Not every generalized word will be found in the memory, especially at the beginning of the story, but these categories still need to be filled. My initial slot-filling implementation mostly filled the categories in randomly. People names were initially pulled from the US Census. The other nouns, which were generalized through WordNet by finding the word's semantic "grandparent", were slot filled by reversing this process. However, since the hierarchy of words branches as it becomes more specific, there are far more options to choose from (e.g. there are many different types of articles of clothing). For now, a noun is selected from this set of possible options randomly. Certain tags were reserved for

pronouns, such as *Synset('male.n.02')* for *he*, so if this Synset was found in the generated sentence, it was filled with its appropriate pronoun. Pronoun and named entity generalization/slot-filling were updated in Section 5.3, and a more sophisticated way of slot-filling will be explained in Section 6.3.2.

**Table 4 – End-to-end examples on previously-unseen input data without slot filling. Let ∅ represent an empty (unfilled) parameter. Examples come from synopses of *Harry Potter and the Prisoner of Azkaban* (a book), *Tales of Monkey Island* (a game; with Guybrush changed to John to be in-vocabulary), and *Carry On Regardless* (a movie), respectively.**

| Experiment | Input | Extracted Event(s) | Generated Next Event(s) | Generated Next Sentence |
|---|---|---|---|---|
| **All Generalized Events & Generalized Sentence** | He reaches out to Remus Lupin, a Defence Against the Dark Arts teacher who is eventually revealed to be a werewolf. | ⟨male.n.02, get-13.5.1, ∅, <PERSON>0⟩<br><br>⟨ORGANIZATION, say-37.7-1, monster.n.01, ∅⟩ | ⟨monster.n.01, amuse-31.1, sarge, ∅⟩<br><br>⟨monster.n.01, amuse-31.1, realize, ∅⟩<br><br>⟨monster.n.01, conjecture-29.5-1, ∅, ∅⟩<br><br>⟨male.n.02, conduit.n.01, entity.n.01, ∅⟩<br><br>⟨male.n.02, free-80-1, ∅, penal institution.n.01⟩ | When nemesis.n.01 dangerous monster.n.01 describes who finally realizes male.n.02 can not, entity.n.01 male.n.02 is released from penal institution.n.01. |
| **Generalized Event & Generalized S+P Sentence** | John unwittingly unleashes an insidious pox that rapidly spreads across the Caribbean. | ⟨<PERSON>0, disassemble-23.3, contagious_disease.n.01, ∅⟩ | ⟨male.n.02, spatial_configuration-47.6, ∅, adopt-93⟩ | male.n.02 crumples and is about to be sheath.n.02. |
| **Original Words Event & S+P Sentence** | He thinks he's on a top secret spying mission. | ⟨he, think, ∅, ∅⟩ | ⟨she, come, know, truth⟩ | She has come to the truth. |

## 3.9 Conclusions

Using a schema—such as the event tuple representation from these experiments—in neural story generation can improve event generation performance, and the type of event representation selected matters. I have created a representation for story events that maintains semantic meaning of textual story data while reducing sparsity of events. The sparsity of events, in particular, results in poor story generation performance. My experiments with different story representations during event-to-event generation support my hypothesis about event representation. I found that the events that most abstracted away from natural language text improve the generative ability of a recurrent neural network's story generation process. Event bigrams did not significantly harm the generative model and will likely help with global consistency as they incorporate more history into the process, although story coherence is difficult and costly to measure and was not evaluated in these experiments. However, the rest of the experiments in the thesis will measure coherence.

Although generalization of events away from natural language appears to help with event successor generation, it poses the problem of making story content unreadable. I introduced a second neural network, event-to-sentence, that learns to translate events with generalized or original words back into natural language. This is important because it is possible for event-to-event to generate events that have never occurred (or have occurred rarely) in a story training corpus. I maintain that being able to recover human-readable

sentences from generalized events is valuable since my event-to-event experiments show use that they are preferred, and it is necessary to be able to fill in specifics later for dynamic storytelling. I presented a pipeline architecture for filling in missing details in automatically generated partially generalized sentences.

Unless stated otherwise, in the rest of this thesis, an event is a tuple $e = \langle wn(s), vn(v), wn(o), wn(m) \rangle$, where $v$ is the verb of the sentence, $s$ is the subject of the verb, $o$ is the object of the verb, and $m$ is a propositional object, indirect object, causal complement, or any other significant noun. The parameters $o$ and $m$ may take the special value of *empty* ($\emptyset$) to denote there is no object of the verb or any additional sentence information, respectively. As explained in Section 3.2, I stem all words and then apply the following functions. The function $wn(\cdot)$ gives the WordNet [78] *Synset* of the argument two levels up in the hypernym tree (i.e. the grandparent *Synset*). The function $vn(\cdot)$ gives the VerbNet [79] class of the argument. See Figure 1 for an example of an "eventified" sentence.

Although using event representations improves the quality of predicting the subsequent sentence in the story (local consistency), there is still the problem of the story plot devolving over time (global consistency). In the next chapter, I discuss the procedure I used to control event generation such that certain plot points are being reached.

# CHAPTER 4. REACHING MAJOR PLOT POINTS

In this chapter, I model story generation as a planning problem: find a sequence of events that transitions the state of the world into one in which the desired *goal* holds. In storytelling, a communicative goal can be to tell a story about a particular domain, to include a theme, or to end the story in a particular way. In the case of this work, the goal is that a given verb (e.g., *marry*, *punish*, *rescue*) occurs in the final event of the story (*Yolanda and Peter get married*)—with the goal being represented as its VerbNet[5] class. While simplistic, it highlights the challenge of control in story generation, especially since it is not enough for the system to jump directly to the desired goal. There are subgoals that need to be reached before the end goal (e.g., in a Western-centric love story, the characters should *meet*, *date*, *love*, etc. before *marry*), and these subgoals usually occur in a certain order, with some variation.

I used reinforcement learning to plan out the events of a story and policy gradients to learn a policy model. I started by training a language model on a corpus of story plots. A language model $P(x_n|x_{n-1}...x_{n-k}; \theta)$ gives a distribution over the possible tokens $x_n$ that are

[5] See Section 3.2 for an explanation of VerbNet and how it is used in my events.

likely to come next given a history of tokens $x_{n-1}...x_{n-k}$ and the parameters of a model $\theta$. This language model is a first approximation of the policy model. Generating a plot by iteratively sampling from this language model, however, provides no guarantee that the plot will arrive at a desired goal except by coincidence. I used REINFORCE [86] to specialize the language model to keep the local coherence of tokens initially learned and also to prefer selecting tokens that move the plot toward a given goal.

Reinforcement learning (RL) addresses some of the issues of preserving coherence for text generation when sampling from a neural language model. It also provides the ability to specify a goal. Reinforcement learning [87] is a technique that is used to solve a *Markov decision process* (MDP). An MDP is a tuple $M = \langle S, A, T, R, \gamma \rangle$ where $S$ is the set of possible world states, $A$ is the set of possible actions, $T$ is a transition function $T$: $S \times A \rightarrow P(S)$, $R$ is a reward function $R$: $S \times A \rightarrow \mathbb{R}$, and $\gamma$ is a discount factor $0 \leq \gamma \leq 1$. The result of reinforcement learning is a *policy* $\pi$: $S \rightarrow A$, which defines which actions should be taken in each state in order to maximize the expected future reward. The *policy gradient* learning approach to reinforcement learning directly optimizes the parameters of a policy model, which is represented as a neural network. One model-free policy gradient approach, REINFORCE [86], learns a policy by sampling from the current policy and backpropagating any reward received through the weights of the policy model. In this work, the actions are events—story points that change the state of the world, and it is worth noting that the transition function between these events is unknown.

Deep reinforcement learning (DRL) has been used successfully for dialog—a similar domain to plot generation. Li et al. [88] pretrained a neural language model and then used a policy gradient technique to reinforce weights that resulted in higher immediate reward for dialogue. They also defined a coherence reward function for their RL agent. Contrasted with task-based dialog generation, story and plot generation often require long-term coherence to be maintained. Thus, I use reward shaping to force the policy gradient search to seek out longer-horizon rewards.

## 4.1 Reinforcement Learning for Plot Generation

If reward is only provided when a generated plot achieves the given goal, then the rewards will be very sparse. Policy gradient learning requires a dense reward signal to provide feedback after every step. As such, I use a reward-shaping technique where the original training corpus is automatically analyzed to construct a dense reward function that guides the plot generator toward the given goal.

### *4.1.1 Initial Language Model*

I split sentences into multiple events, where possible, which creates a potential one-to-many relationship between an original sentence and the event(s) produced from it. However, once produced, the events are treated sequentially.

In this experiment, I use an encoder-decoder network [70] as my starting language model and my baseline, such as the ones used in Section 3.3. Encoder-decoder networks

can be trained to generate sequences of text for dialogue or story generation by pairing one or more sentences in a corpus with the successive sentence and learning a set of weights that captures the relationship between the sentences. My language model is thus $P(e_{i+1}|e_i ; \theta)$ where $e_i = \langle s_i, v_i, o_i, m_i \rangle$, using the original 4-tuple event representation introduced in Section 3.2.

I seek a set of policy model parameters $\theta$ such that $P(e_{i+1}|e_i ; \theta)$ is the distribution over events according to the corpus *and* also that increase the likelihood of reaching a given goal event in the future. For each input event $e_i$ in the corpus, an action involves choosing the most probable next event. The reward is calculated by determining how far the event $e_{i+1}$ is from my given goal event. The final gradient used for updating the parameters of the network and shifting the distribution of the language model is calculated as follows:

$$\nabla_\theta J(\theta) = r_{i+1} \nabla_\theta \log P(e_{i+1}|e_i;\theta) \tag{4}$$

where $e_{i+1}$ and $R(v(e_{i+1}))$ are, respectively, the event chosen at timestep $i+1$ and the reward for the verb in that event. The policy gradient technique thus gives an advantage to highly-rewarding events by facilitating a larger step towards the likelihood of predicting these events in the future, over events which have a lower reward. In the next section I describe how $R(v(e_{i+1}))$ is computed.

## 4.2 Reward Shaping

For the purpose of *controllability* in plot generation, I wish to reward the network whenever it generates an event that makes it more likely to achieve the given goal. Here, the goal is a given VerbNet class that I wish to see at the end of a plot. *Reward shaping* [89] is a technique whereby sparse rewards—such as rewarding the agent only when a given goal is reached—are replaced with a dense reward signal that provides rewards at intermediate states in the exploration leading to the goal.

To produce a smooth, dense reward function, I observed that certain events—and thus certain verbs—are more likely to appear closer to the goal than others in story plots. For example, suppose my goal is to generate a plot in which one character *admires* another (*admire-31.2* is the VerbNet class that encapsulates the concept of falling in love). Events that contain the verb *meet* are more likely to appear nearby events that contain *admire*, whereas events that contain the verb *leave* are likely to appear farther away. To construct the reward function, I pre-process the stories in my training corpus and calculate two key components: (a) the distance of each verb from the target/goal verb, and (b) the frequency of the verbs found in existing stories.

### *4.2.1 Distance*

The distance component of the reward function measures how close the verb *v* of an event is to the target/goal verb *g*, which is used to reward the model when it produces events with

verbs that are closer to the target verb. The formula for estimating this metric for a verb *v* is:

$$r_1(v) = \log \sum_{s \in S_{v,g}} l_s - d_s\,(v, g) \tag{5}$$

where $S_{v,g}$ is the subset of stories in the corpus that contain *v* prior to the goal verb *g*, $l_s$ is the length of story *s*, and $d_s(v,g)$ is the number of events between the event containing *v* and the event containing *g* in story *s* (i.e., the distance within a story). Subtracting from the length of the story produces a larger reward when events with *v* and *g* are closer.

### *4.2.2 Story-Verb Frequency*

Story-verb frequency rewards based on how likely any verb is to occur in a story before the target verb. This component estimates how often a particular verb *v* appears before the target verb *g* throughout the stories in the corpus. This discourages the model from outputting events with verbs that rarely occur in stories before the target verb. The following equation is used for calculating the story-verb frequency metric:

$$r_2(v) = \log \frac{k_{v,g}}{N_v} \tag{6}$$

where $N_v$ is the number of times verb *v* appears throughout the corpus, and $k_{v,g}$ is the number of times *v* appears before the goal verb *g* in any story.

### *4.2.3 Final Reward*

The final reward for a verb—and thus the event as a whole—is calculated as the product of the distance and frequency metrics. The rewards are normalized across all the verbs in the corpus and the final reward is:

$$R(v) = \alpha \times r_1(v) \times r_2(v) \quad (7)$$

where α is the normalization constant.

When combined, both *r* metrics advantage verbs that 1) appear close to the target, while also 2) being present before the target in a story frequently enough to be considered significant.

### *4.2.4 Verb Clustering*

As mentioned in the introduction to this chapter, there is no way of knowing how to transition from one event to another, except what the initial sequence-to-sequence network learned from the data. This means that the model can transition from any one event to any other event at any point. In order to discourage the model from jumping to the target quickly, I clustered my entire set of verbs based on Equation 7 using the Jenks Natural Breaks optimization technique [90]. I restrict the vocabulary of $v_{out}$—the model's output verb—to the set of verbs in the $c+1^{th}$ cluster, where *c* is the index of the cluster that verb $v_{in}$—the model's input verb—belongs to. In these experiments, the verbs were divided into

30 clusters. The rest of the event is generated by sampling from the full distribution. The intuition is that by restricting the vocabulary of the output verb, the gradient update in

$$\nabla_\theta J(\theta) = r_{i+1} \nabla_\theta \log P(e_{i+1}|e_i;\theta) \quad (8)$$

takes greater steps toward verbs that are more likely to occur next (i.e., in the next cluster) in a story headed toward a given goal. If the sampled verb has a low probability, the step will be smaller than in the situation in which the verb is highly probable according to the language model.

## 4.3 Automated Evaluation

I ran experiments to measure three properties of my model: (1) how often the model can produce a plot—a sequence of events—that contains a desired target verb; (2) the perplexity of the model; and (3) the average length of the stories. Perplexity is a measure of the predictive ability of a model; particularly, how "surprised" the model is by occurrences in a corpus. I compare my results to those of a baseline event-to-event story generation model from Section 3.3.

### *4.3.1 Corpus Preparation*

I use the CMU movie summary corpus [71]. However, this corpus proved to be too diverse; there is high variance between stories, which dilutes event patterns. I used Latent Dirichlet Analysis to cluster the stories from the corpus into 100 "genres". I selected a cluster that

appeared to contain soap-opera--like plots. The stories were "eventified"—turned into *event* sequences, as explained in Section 3.2. I chose *admire-31.2* and *marry-36.2* as two target verbs because those VerbNet classes capture the sentiments of "falling in love" and "getting married", which are appropriate for my sub-corpus. The romance corpus was split into 90% training, and 10% testing data. I used consecutive events from the eventified corpus as source and target data, respectively, for training the sequence-to-sequence network.

### *4.3.2 Model Training*

For my experiments I trained the encoder-decoder network using Tensorflow. Both the encoder and the decoder comprised of LSTM units, with a hidden layer size of 1024. The network was pre-trained for a total of 200 epochs using mini-batch gradient descent and batch size of 64.

I created three models:

- *Seq2Seq*: This pre-trained model is identical to the "generalized multiple sequential event-to-event" model from Section 3.3. This is my baseline.
- *DRL-clustered*: Starting with the weights from the *Seq2Seq*, I continued training using the policy gradient technique and the reward function, along with the clustering and vocabulary restriction in the verb position described in the previous section, while keeping all network parameters constant.

- *DRL-unrestricted*: This is the same as *DRL-clustered* but without vocabulary restriction while sampling the verb for the next event during training (§4.2.4).

*DRL-unrestricted* assigns a reward for each verb transition without any vocabulary restriction. The DRL-clustered and DRL-unrestricted models are trained for a further 200 epochs than the baseline.

### *4.3.3 Experimental Setup*

With each event in my held-out dataset as a seed event, I generated stories with my baseline Seq2Seq, DRL-clustered, and DRL-unrestricted models. For all models, the story generation process was terminated when either:

(1) the model outputs an event with the target verb;

(2) the model outputs an end-of-story token; or

(3) the length of the story reaches 15 lines.

The goal achievement rate was calculated by measuring the percentage of these stories that ended in the target verb (*admire* or *marry*). Additionally, I noted the length of generated stories and compare it to the average length of stories in my test data where the goal event occurs (assigning a value of 15 if it does not occur at all). Finally, I measure the perplexity for all the models. Since the testing data is not a model, perplexity cannot be calculated.

**Table 5 – Results of the automated experiments, comparing the goal achievement rate, average perplexity, and average story length for the testing corpus, baseline Seq2Seq model, and my clustered and unrestricted DRL models.**

| Goal | Model | Goal Achievement Rate | Average Perplexity | Average Story Length |
|---|---|---|---|---|
| **admire** | Test Corpus | 20.30% | n/a | 7.59 |
| | Seq2Seq | 35.52% | 48.06 | 7.11 |
| | Unrestricted | 15.82% | **5.73** | 7.32 |
| | Clustered | **94.29%** | 7.61 | 4.90 |
| **marry** | Test Corpus | 24.64% | n/a | 7.37 |
| | Seq2Seq | 39.92% | 48.06 | 6.94 |
| | Unrestricted | 24.05% | 9.78 | 7.38 |
| | Clustered | **93.35%** | **7.05** | 5.76 |

### *4.3.4 Results and Discussion*

Results are summarized in Table 5. Only 22.47% of the stories in the testing set, on average, end in my desired goals, illustrating how rare the chosen goals were in the corpus. The DRL-clustered model generated the given goals on average 93.82% of the time, compared to 37.72% on average for the baseline Seq2Seq and 19.935% for the DRL-unrestricted model. This shows that my policy gradient approach can direct the plot to a pre-specified ending and that my clustering method is integral to doing so. Removing verb clustering from my reward calculation to create the DRL-unrestricted model harms goal achievement; the system rarely sees a verb in the next cluster so the reward is frequently low, making distribution shaping towards the goal difficult.

I use perplexity as a metric to estimate how similar the learned distribution is for accurately predicting unseen data. I observe that perplexity values drop substantially for

the DRL models (7.61 for DRL-clustered and 5.73 for DRL-unrestricted with goal *admire*; 7.05 for DRL-clustered and 9.78 for DRL-unrestricted with goal *marry*) when compared with the Seq2Seq baseline (48.06). This can be attributed to the fact that my reward function is based on the distribution of verbs in the story corpus and thus is refining the model's ability to recreate the corpus distribution. Because DRL-unrestricted's rewards are based on subsequent verbs in the corpus instead of verb clusters, it sometimes results in a lower perplexity, but at the expense of not learning how to achieve the given goal too often.

The average story length is an important metric because it is trivial to train a language model that reaches the goal event in a single step. It is not essential that the DRL models produce stories approximately the same length as the those in the testing corpus, as long as the length is not extremely short (leaping to conclusions) or too long (the story generator is timing out). The baseline Seq2Seq model creates stories that are about the same length as the testing corpus stories, showing that the model is mostly mimicking the behavior of the corpus it was trained on. The DRL-unrestricted model produces similar behavior, due to the absence of clustering or vocabulary restriction to prevent the story from rambling. However, the DRL-clustered model creates slightly shorter stories, showing that it is reaching the goal quicker, while not jumping immediately to the goal. Doing the verb clustering and resampling alleviates the issue of lacking a transition function, but it is important to point out that this solution does not completely replace a transition function since the model can still generate any two events together at decode time; it only becomes less likely to do so.

## 4.4 Human Evaluation

The best practice in the evaluation of story and plot generation is human subject evaluation. However, the use of the event tuple representation makes human subject evaluation difficult because events are not easily readable. In Section 3.5, I used a second neural network to translate events into human-readable sentences, but my technique did not have sufficient accuracy to use in a human evaluation, particularly on generated events. The use of a second network also makes it impossible to isolate the generation of events from the generation of the final natural language sentence when it comes to human perception. To overcome this challenge, I have developed an evaluation protocol that allows me to directly evaluate plots with human judges. Specifically, I recruited and taught individuals to convert event sequences into natural language before giving generated plots to human judges. By having concise, grammatically- and semantically-guaranteed human translations of generated plot events I know that the human judges are evaluating the raw events and not the creative aspects of the way sentences are written.

### *4.4.1 Corpus Creation*

I randomly selected 5 stories generated by my DRL-clustered system, 5 stories generated by the Seq2Seq baseline system, and 3 from the eventified testing corpus. I trained 26 people unassociated with the research team to "translate" events into short natural language sentences. The testing corpus was mainly used to verify that the translation process was

accurate since I do not expect my models to reach this upper bound; thus only three stories were selected.

Each translator was instructed that their "primary goal is to translate stories from an abstract 'event' representation into a natural language sentence." The instructions then continued with:

(1) a refresher on parts of speech,

(2) the format of the event representation,

(3) examples of events and their corresponding sentences,

(4) resources on WordNet and VerbNet with details on how to use both, and

(5) additional general guidelines and unusual cases they might encounter (e.g., how to handle *empty* parameters in events).

The translators were further instructed to try to produce sentences that were as faithful to the event as possible and to not add extraneous details, swap the order of words in the event, nor choose a better verb even if the plot would be improved. The full instructions can be found here.

Pairs of people translated plots individually and then came together to reach a consensus on a final version of the plot. That is, human translators reversed the eventification process shown in Figure 1 to create a human readable sentence from an event. Table 6 shows an example of an entire eventified story and the corresponding human

translations. The full set of stories used in the final experiment can be found in Section A.1 of the Appendix.

**Table 6 – An example of an eventified story and the translation written by a pair of participants. NE refers to a named entity; *Ø* refers to an empty parameter.**

| **DRL Event Output** ⟨ ***subject, verb, object, modifier*** ⟩ | **Translated Sentence** |
|---|---|
| ⟨ relative.n.01, disappearance-48.2, *Ø*, *Ø* ⟩ | My cousin died. |
| ⟨ NE1, say-37.7-1, visit, *Ø* ⟩ | Alexander insisted on a visit. |
| ⟨ NE1, meet-36.3-1, female.n.02, *Ø* ⟩ | Alexander met her. |
| ⟨ NE0, correspond-36.1, *Ø*, NE1 ⟩ | Barbara commiserated with Alexander. |
| ⟨ physical_entity.n.01, marry-36.2, *Ø*, *Ø* ⟩ | They hugged. |
| ⟨ group.n.01, contribute-13.2-2, *Ø*, LOCATION ⟩ | The gathering dispersed to Hawaii. |
| ⟨ gathering.n.01, characterize-29.2-1-1, time_interval.n.01, *Ø* ⟩ | The community remembered their trip. |
| ⟨ physical_entity.n.01, cheat-10.6, pack, *Ø* ⟩ | They robbed the pack. |
| ⟨ physical_entity.n.01, admire-31.2, social_gathering.n.01, *Ø* ⟩ | They adored the party. |

### *4.4.2 Experimental Setup*

I recruited 175 participants on Amazon Mechanical Turk using TurkPrime [91]. Each participant was compensated $10 for completing the questionnaire. To qualify, participants needed to answer a question about what is “most closely related to a soap opera” with the choices of musical, drama, or comedy—the answer being drama. If the participant got this question wrong, they were not permitted to take the survey. Participants were given one of the translated plots at a time, rating each of the following statements on a 5-point Likert

scale for how much they agreed (Strongly Agree, Somewhat Agree, Neither Agree nor Disagree, Somewhat Disagree, or Strongly Disagree):

1. This story exhibits CORRECT GRAMMAR.
2. This story's events occur in a PLAUSIBLE ORDER.
3. This story's sentences MAKE SENSE given sentences before and after them.
4. This story AVOIDS REPETITION.
5. This story uses INTERESTING LANGUAGE.
6. This story is of HIGH QUALITY.
7. This story is ENJOYABLE.
8. This story REMINDS ME OF A SOAP OPERA.
9. This story FOLLOWS A SINGLE PLOT.

By making the equal interval assumption, I can turn the Likert values into numerals from 1 for Strongly Disagree to 5 for Strongly Agree.

The first seven questions are taken from a tool designed specifically for the evaluation of computer-generated stories that has been validated against human judgements [92]. Each participant answered the questions for all three story conditions. The question (#8) about the story being a soap opera was added to determine how the performance of the DRL story generator affects reader perceptions of the theme, since the system was trained on soap-opera—like plots. The single plot question (#9) was added to determine if my DRL model was maintaining the plot better than the Seq2Seq model. The questions about correct

grammar, interesting language, and avoiding repetition are irrelevant to my evaluation since the natural language was produced by the human translators, but were kept for consistency with Purdy et al [92]. Finally, participants answered two additional questions that required short answers to be typed:

1. Please give a summary of the story above in your own words.
2. For THIS STORY, please select which of the previous attributes (e.g. enjoyable, plausible, coherent) you found to be the MOST IMPORTANT and explain WHY.

The answers to these questions were not evaluated. Instead, if any participants failed to answer the short answer questions, their data was removed from the results. Twenty-five participants were removed in total.

### *4.4.3 Results and Discussion*

I performed one-way repeated-measures ANOVA with post-hoc Tukey HSD on the collected data since each participant rated a story from each category. Average scores and their significance across conditions can be seen in Figure 4. Questions on interesting language and avoiding repetition are not found to be significant across all three conditions.

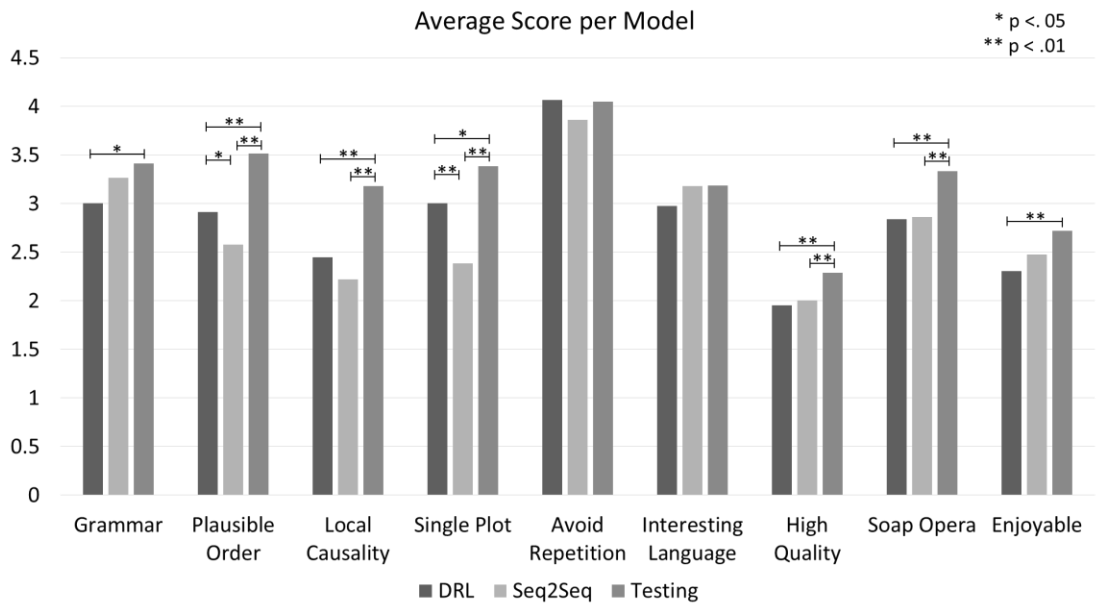


**Figure 4 – Amazon Mechanical Turk participants rated a story from each category across nine different dimensions on a scale of 1-Strongly Disagree to 5-Strongly Agree. Single asterisks stand for p<0.05. Double asterisks stand for p<0.01.**

Since these are not related to event generation model performance this provides an indication that the translations are fair across all conditions. Grammar was significantly different between testing corpus stories and DRL-generated stories $p<0.05$, which was unanticipated. Upon further analysis, both the baseline Seq2Seq model and the DRL model generated *empty* values for the object and modifier at higher rates than found in the corpus. It is harder to make complete, grammatical sentences with only two tokens in an event, namely when a verb is transitive--requiring at least one object. Beyond more expected results such as having a better plausible order, the testing corpus stories were also significantly more likely to be perceived as being soap operas $p<0.01$, the genre from which the corpus stories were drawn. It is unclear why this would be the case, except that

both the Seq2Seq and DRL models could be failing to learn some aspect of the genre despite being trained on the same corpus.

Stories in the DRL condition were significantly perceived to have more plausible orderings than those in the baseline Seq2Seq condition $p<0.05$ and were significantly more likely to be perceived as following a single plot $p<0.05$. Since stories generated by the baseline Seq2Seq model begin to lose coherence as the story progresses, these results confirm my hypothesis that the DRL's use of reward shaping keeps the plot on track. The DRL is also perceived as generating stories with more local causality than the Seq2Seq, although the results were not statistically significant. For all other dimensions, the DRL stories are not found to be significantly different than baseline stories. When continuing the training of a pre-trained language model using a reward function instead of the standard cross entropy loss there is a non-trivial chance that model updates will degrade any aspect of the model that is not related to goal achievement. Thus, a positive result is one in which DRL-condition stories are never significantly lower than Seq2Seq-condition stories. This shows that I am able to get to the goal state without any significant degradation in the other aspects of story generation compared to a baseline.

### 4.5 Next Steps: Improving Sentence Generation and Addressing Global Coherence

This work shows that a language model trained on a particular corpus can be manipulated to reach certain goals using the same corpus, it would be premature to say that automated storytelling is “solved”. There are many aspects of this system that can be improved, but I

will focus specifically on two that push the system to, respectively, have more relevant & interesting sentences generated from events and have a causal ordering of events.

Independently-trained sequence-to-sequence models have a hard time "understanding" each other. When output from the event-to-event model is often unlike anything the event-to-sentence model was trained on, and so the event-to-sentence model subsequently spits out unintelligible, irrelevant sentences. On the other hand, if it is something the event-to-sentence model has seen before, the output often ends up being boring and predictable. In the next chapter (CHAPTER 5), I will share the details of an ensemble method I created to strike this balance between relevant and interesting sentence generation.

Another pain point is that the current system is lacking any sort of understanding of the real world. Usually consequences of actions—even in stories—are limited by physical, temporal, or societal constraints. The system at this state lacks this kind of commonsense understanding of the world and thus results in many unreasonable sequences of events. In CHAPTER 6, I take a look at neuro-symbolic and symbolic systems for story generation that take into consideration causal links between events/sentences in a story, comparing them to neural-only models.

# CHAPTER 5. IMPROVING EVENT-TO-SENTENCE

For neural-based approaches to story and plot generation, a neural language model is trained on a corpus of stories to predict the next character, word, or sentence in a sequence based on a history of tokens [58], [60], [93]–[96]. The advantage of neural-based approaches is that there is no need for explicit domain modeling beyond providing a corpus of example stories. The primary pitfall of neural language model approaches for story generation is that the space of stories that can be generated is huge, which in turn, implies that, in a textual story corpora, any given sentence will likely only be seen once.

In this chapter and the following chapter, I am using a modified version of my event representation, adding an argument for a preposition. Thus, my new event representation becomes ⟨ *s*, *v*, o, *p*, *m* ⟩, where *p* is the preposition and the following are still the same: *s* is the subject, *v* is the verb, *o* is the direct object, and *m* is the modifier noun, indirect object, or object of the prepositional phrase for which the preposition *p* heads. Using my event representation (Section 3.2), I can reduce the sparsity in a dataset that comes from an abundance of unique sentences [58]. The event representation enables the decomposition of the plot generation task into two sub-problems: *event-to-event* (Section 3.3) and *event-to-sentence* (Section 3.5). The event-to-sentence problem can be thought of as a translation task−translating from the language of events into natural language. I found [58], however, that independently-trained sequence-to-sequence LSTM networks [70] frequently ignore the input event and only generate text based on the original corpus, overwriting the plot-

based decisions made during event-to-event. There are two contributing factors. Firstly, event-to-event models tend to produce previously-unseen events, which, when fed into the event-to-sentence model result in unpredictable behavior. A basic sequence-to-sequence model is unable to learn how to map these unseen events to sentences. Secondly, sentences are often only seen once in the entire corpus. Despite the conversion into events, the sparsity of the data means that each event is still likely seen a limited number of times. For these reasons, I framed the event-to-sentence task as *guided language generation*, using a generated event as a guide. In this chapter, I present an ensemble-based system for the event-to-sentence problem that balances between retaining the event's original semantic meaning, while being an interesting continuation of the story. I demonstrate that my system for guided language generation outperforms a baseline sequence-to-sequence approach. Additionally, I present the results of the full end-to-end story generation pipeline (Figure 2), showing how the system performs when all components are integrated.

### 5.1 Science Fiction: A New Corpus

To aid in the performance of my story generation, I select a single genre: science fiction. I scraped a collection of long-running science fiction TV show plot summaries from the fandom wiki service *wikia.com*. This dataset contains longer and more detailed plot summaries than the dataset used in my previous work [58], [97], which I believe to be important for the overall story generation process. Although this dataset has significantly fewer stories than the movie corpus (2,276 stories in total, each story an episode of a TV

show; compared to the 42,170 stories in the movie corpus), it is far more focused and the stories are usually much longer. The average story length is 89.23 sentences. There are stories from 11 shows, with an average of 207 stories per show, from shows like *Doctor Who*, *Futurama*, and *The X-Files*. The data was pre-processed to simplify alien names in order to aid the parser. Then the sentences were split, partially following the "split-and-pruned" methodology of my work as described in Section 3.5 [58].

Once the sentences were split, they were "eventified" as described in Section 3.3. One benefit of having split sentences is that there is a higher chance of having a 1:1 correspondence between a sentence and an event, instead of a single sentence becoming multiple events. After the data is fully prepared, it is split in an 8:1:1 ratio to create the training, validation, and testing sets, respectively. This corpus is used in all of the following experiments of the thesis.

## 5.2 The Event-to-Sentence Ensemble

I define event-to-sentence to be the problem of selecting a sequence of words $s_t = s_{t_0}, s_{t_1}, \ldots, s_{t_k}$−that form a sentence−given the current input event $e_t$, i.e. the current sentence is generated based on maximizing $Pr(s_t|e_t; \theta)$ where $\theta$ refers to the parameters of the generative system. The eventification in Section 3.3 is a lossy process in which some of the information from the original sentence is dropped. Thus, the task of event-to-sentence involves filling in this missing information. There is also no guarantee that the event-to-event process will produce an event that is part of the event-to-sentence training

corpus, simply due to the fact that the space of potentially-generated events is very large; the correct mapping from the generated event to a natural language sentence would be unknown.

In prior work [58], I use a sequence-to-sequence LSTM neural network to translate events into sentences. I observe that "vanilla" sequence-to-sequence networks end up operating as simple language models, often ignoring the input event when generating a sentence. The generated sentence is usually grammatically correct but retains little of the semantic meaning given by the event.

I thus look for other forms of guided neural language generation, with the goals of preserving the semantic meaning from the event in addition to keeping the generated sentences interesting. I propose four different models−optimized towards a different point in the spectrum between the two objectives, and a baseline fifth model that is used as a fall-through. The task of each model is to translate events into "generalized" sentences, wherein nouns are replaced by WordNet Synsets. If a model does not pass a specific threshold (determined individually for each model), the system continues onto the next model in the ensemble. In order, the models are: (1) a retrieve-and-edit model based on Hashimoto et al. [98]; (2) template filling; (3) sequence-to-sequence with Monte Carlo beam decoding; (4) sequence-to-sequence with a finite state machine decoder; and (5) vanilla (beam-decoding) sequence-to-sequence. I find that none of these models by themselves can successfully find a balance between the goals of *retaining all of the event tokens* and

*generating interesting output*. However, each of the models possess their own strengths and weaknesses−each model is essentially optimized towards a different point on the spectrum between the two goals. I combine these models into an ensemble in an attempt to minimize the weaknesses of each individual model and to achieve a balance.

### *5.2.1 Retrieve-and-Edit*

The first model is based on the retrieve-and-edit (*RetEdit*) framework for predicting structured outputs [98]. I first learn a task-specific similarity between event tuples by training an encoder-decoder to map each event onto an embedding that can reconstruct the output sentence; this is my retriever model. Next, I train an editor model which maximizes the likelihood of generating the target sentence given both the input event and a retrieved event-sentence example pair. I used a standard sequence-to-sequence model with attention and copying [99] to stand in as my editor architecture. Although this framework was initially applied to the generation of GitHub Python code and Hearthstone cards, I extend this technique to generate sentences from my event tuples. Specifically, I first initialize a new set of GLoVe word embeddings [100], using random initialization for out-of-vocabulary words. I use my training set to learn weights for the retriever and editor models, set confidence thresholds for the model with the validation set, and evaluate performance using the test set.

In order to generate a sentence from a given input event, there are two key phases: “retrieve” phase and “edit” phase. With respect to the input event, I first retrieve the

nearest-neighbor event and its corresponding sentence in the training set using the retriever model. Passing both the retrieved event-sentence pair and the input event as inputs, I use the editor model to generate a sentence using beam search.

Many of the successes produced by the model stem from its ability to retain the complex sentence structures that appear in my training corpus and thus attempts to balance between maintaining coherence and being interesting. However, this interaction with the training data can also prove to be a major drawback of the method; target events that are distant in the embedding space from training examples typically result in poor sentence quality. Since RetEdit relies heavily on having good examples, I set the confidence of the retrieve-and-edit model to be proportional to $1-$*retrieval distance* when generating sentences, as a lower retrieval distance implies greater confidence. However, the mapping from event to sentence is not a one-to-one function. There are occasionally multiple sentences that map to a single event, resulting in retrieval distance of 0, in which case the example sentence is returned without modifications.

### *5.2.2 Sentence Templating*

As mentioned earlier, the baseline sequence-to-sequence network operates as a simple language model and can often ignore the input event when generating a sentence. However, I know that my inputs, an event tuple will have known parts of speech. I created a simplified grammar for the syntax of sentences generated from events:

$$
\begin{aligned}
S &\rightarrow NP\ v\ (NP)\ (PP) \\
NP &\rightarrow d\ n \\
PP &\rightarrow p\ NP
\end{aligned}
\tag{9}
$$

where *d* is a determiner that will be added and the rest of the terminal symbols correspond to an argument in the event, with *n* being *s*, *o*, or *m*, depending on its position in the sentence. The resulting sentence would be [ _ s] {v [ _ o] [p _ m]} where underscores indicate where words should be added to make a complete sentence.

First, my algorithm predicts the most likely VerbNet frame based on the contents of the input event (how many and which arguments are filled). VerbNet provides a number of syntactic structures for different verb classes based on how the verb is being used. For example, if the input event contains two nouns and a verb without a preposition, I assume that the output sentence takes the form of [NP V NP], but if it has two nouns, a verb, *and* a proposition, then it should be [NP V **PP**].

Second, I apply a Bidirectional LSTM language model trained on the generalized sentences in my training corpus. Given a word, I can generate words before and after it, within a particular phrase as given by some of the rules above, and concatenate the generated sentence fragments together. Specifically, I use the AWD-LSTM [101] architecture as my language model since it is currently state-of-the-art.

At decode time, I continue to generate words in each phrase until I reach a stopping condition: (1) reaching a maximum length (to prevent run-on sentences); or (2) generating a token that is indicative of an element in the next phrase, for example seeing a verb being generated in a noun phrase. When picking words from the language model, I noticed that stop words like “the” and “and” were extremely common. To increase the variety of the sentences, I sampled from the top *k* most-likely next words and enforced a number of grammar-related rules in order to keep the coherence of the sentence. For example, I did not allow two determiners nor two nouns to be generated next to each other.

One can expect that many of the results will look structurally similar. However, I can guarantee that the provided tokens in the event will appear in the generated sentence−this model is optimized towards maintaining coherence. To determine the confidence of the model for each sentence, I sum the loss after each generated token, normalize to sentence length, and subtract from 1 as higher loss translates to lower confidence.

### *5.2.3 Monte-Carlo Beam Search*

My third method is an adaptation of *Monte Carlo Beam Search* [102] for event-to-sentence. I train a sequence-to-sequence model on pairs of events & generalized sentences and run Monte Carlo beam search at decode time. This method differs from traditional beam search in that it introduces another scoring term that is used to re-weight all the beams at each timestep.

After top-scoring words are outputted by the model at each timestep, playouts are done from each word, or *node*. A node is the final token of the partially-generated sequences on the beam currently and the start of a new playout. During each playout, one word is sampled from the current step's softmax over all words in the vocabulary. The decoder network is unrolled until it reaches the "end-of-story" tag. Then, the previously-generated sequence and the sequence generated from the current playout are concatenated together and passed into a scoring function that computes the current playout's score.

The scoring function is a combination of (1) BLEU scores up to 4-grams between the input event and generated sentence, as well as (2) a weighted 1-gram BLEU score between each item in the input event and generated sentence. The weights combining the 1-gram BLEU scores are learned during validation time where the weight for each word in the event that *does not* appear in the final generated sequence gets bumped up. Multiple playouts are done from each word and the score *s* for the current word is computed as:

$$s_t = \alpha * s_{t-1} + (1 - \alpha) * AVG(playout_t) \quad (10)$$

where α is a constant.

In the end, the *k* partial sequences with the highest playout scores are kept as the current beam. For the ensemble, this model's confidence score is the final score of the highest-scoring end node. Monte Carlo beam search excels at creating diverse output−i.e. it skews towards generating interesting sentences. Since the score for each word is based on

playouts that sample based on weights at each timestep, it is possible for the output to be different across runs. The Monte Carlo beam decoder has been shown to generate better sentences that are more grammatically-correct than the other techniques in my ensemble, while sticking more to the input than a traditional beam decoder. However, there is no guarantee that all input event tokens will be included in the final output sentence.

### *5.2.4 Finite State Machine Constrained Beams*

Various forms of beam search, including Monte Carlo playouts, cannot ensure that the tokens from an input event appear in the outputted sentence. As such, I adapted the algorithm to fit such lexical constraints, similar to Anderson et al. [103] who adapted beam search to fit captions for images, with the lexical constraints coming from sets of image tags. The *Constrained Beam Search* used finite state machines to guide the beam search toward generating the desired tokens. Their approach, which I have co-opted for event-to-sentence, attempts to achieve a balance between the flexibility and sentence quality typical of a beam search approach, while also adhering to the context and story encoded in the input events that more direct approaches (e.g. templates, *Section 5.2.2*) would achieve.

The algorithm works on a per-event basis, beginning by generating a finite state machine. This finite state machine consists of states that enforce the presence of input tokens in the generated sentence. As an example, assume I have an *n*-token input event, $\{t_1, t_2, t_3, \dots, t_n\}$. The corresponding machine consists of $2^{\mathrm{n}}$ states. Each state maintains a search beam of size $B^s$ with at most *b* output sequences, corresponding to the configured

beam size *s*. At each time step, every state (barring the initial state) receives from predecessor states those output sequences whose last generated token matches an input event token. The state then adds to its beam the *b* most likely output sequences from those received. Generating token $t_1$ moves the current state from the initial state to the state corresponding to $t_1$, $t_3$ to a state for $t_3$, and so on. The states $t_1$ and $t_3$ then, after generating tokens $t_1$ and $t_3$ respectively, transmit said sequences to the state $t_{1,3}$. The states and transitions proceed as such until reaching the final state, wherein they have matched every token in the input event. Completed sequences in the final state contain all input event tokens, thus providing the ability to retain the semantic meaning of the event.

As much as the algorithm is based around balancing generating good sentences with satisfying lexical constraints, it does not perform particularly well at either. It is entirely possible, if not at all frequent, for generated sentences to contain all input tokens but lose proper grammar and syntax, or even fail to reach the final state within a fixed time horizon. This is exacerbated by larger tuples of tokens, seen even at just five tokens per tuple. To compensate, I relax my constraint to permit output sequences that have matched at least three out of five tokens from the input event.

### *5.2.5 Ensemble*

The entire event-to-sentence ensemble is designed as a cascading sequence of models: (1) retrieve-and-edit, (2) sentence templating, (3) Monte Carlo beam search, (4) finite state constrained beam search, and (5) standard beam search. I use the confidence scores

generated by each of the models in order to re-rank the outputs of the individual models. This is done by setting a confidence threshold for each of the models such that if a confidence threshold fails, the next model in the ensemble is tried. The thresholds are tuned on the confidence scores generated from the individual models on the validation set of the corpus. This ensemble saves on computation as it sequentially queries each model, terminating early and returning an output sentence if the confidence threshold for any of the individual models are met.

An event first goes through the retrieve-and-edit framework, which generates a sentence and corresponding confidence score. This framework performs well when it is able to retrieve a sample from the training set that is relatively close in terms of retrieval distance to the input. Given the sparsity of the dataset, this happens with a relatively low probability, and so I place this model first in the sequence.

The next two models are each optimized towards one of my two main goals. The sentence templating approach retains all of the tokens within the event and so loses none of its semantic meaning, at the expense of generating a more interesting sentence. The Monte-Carlo approach, on the other hand, makes no guarantees regarding retaining the original tokens within the event but is capable of generating a diverse set of sentences. I thus cascade first to the sentence templating model and then the Monte-Carlo approach, implicitly placing greater importance on the goal of retaining the semantic meaning of the event.

The final model queried is the finite-state-machine−constrained beam search. This model has no confidence score; either the model is successful in producing a sentence within the given length with the event tokens or not. In the case that the finite state machine-based model is unsuccessful in producing a sentence, the final fall-through model−the baseline sequence-to-sequence model with standard beam search decoding−is used.

## 5.3 Experiments

I perform two sets of experiments, one set evaluating my models on the event-to-sentence problem by itself, and another set intended to evaluate the full storytelling pipeline. Each of the models in the event-to-sentence ensemble are trained on the training set in the sci-fi corpus. The training details for each of the models are as described above. All of the models in the ensemble slot-fill the verb automatically—filling a VerbNet class with a verb of appropriate conjugation—except for the sentence templating model which does verb slot-filling during post-processing.

After the models are trained, I pick the cascading thresholds for the ensemble by running the validation set through each of the models and generating confidence scores. This is done by running a grid search through a limited set of thresholds such that the overall BLEU-4 score [80] of the generated sentences in the validation set is maximized. These thresholds are then frozen when running the final set of evaluations on the test set. For the baseline sequence-to-sequence method, I decode my output with a beam size of 5.

I report perplexity, BLEU-4, and ROUGE-4 scores, comparing against the gold standard from the test set.

$$Perplexity = 2^{-\sum_x p(x) \log_2 p(x)} \tag{11}$$

where *x* is a token in the text, and

$$p(x) = \frac{count(x)}{\sum_{v \in V} count(V)} \tag{12}$$

where *V* is the vocabulary. My BLEU-4 scores are naturally low (where higher is better) because of the creative nature of the task—good sentences may not use any of the ground-truth *n*-grams. Even though I frame *event-to-sentence* as a translation task, BLEU-4 and ROUGE-4 are not reliable metrics for creative generation tasks.

The first experiment takes plots in the in the test set, eventifies them, and then uses my *event-to-sentence* ensemble to convert them back to sentences. In addition to using the full ensemble, I further experiment with using different combinations of models along the spectrum between maintaining coherence and being interesting. I then evaluate the generated sentences, using the original sentences from the test set as a gold standard.

The second experiment uses event sequences generated by an event-to-event system such as my controllable work [97] and is designed to demonstrate how my system integrates into the larger pipeline illustrated in Figure 2. I then transform these generated

event sequences into generalized sentences using both the ensemble and the baseline sequence-to-sequence approach. As the last step, the generalized sentences are passed into the "slot filler" such that the categories are filled. As the story goes on, the "memory" maintains a dynamic graph that keeps track of what entities (e.g. people, items) are mentioned at which event and what their tag (e.g. <PERSON>5, Synset('instrument.n.01')) was. When the slot filler sees a new sentence, it first tries to see if it can fill it in with an entity it as seen before. This includes if the current Synset it is looking at is a descendent of a Synset already stored in memory. If a new word has to be selected, named entities are randomly chosen from a list collected from the original science fiction corpus, with entities paired with their respective tags (PERSON, ORGANIZATION, NUMBER, etc.). Synsets are selected by finding a descendent 1 or 2 levels down. The word is currently selected randomly, but this will soon be improved by the addition of a language model guiding it. To fill a pronoun (<PRP>), the slot filler refers to the memory to select a recently-mentioned entity. Person names are run through US Census data to determine the "gender" of the name in order to select an appropriate pronoun. If no pronoun can be found, it defaults to *they*/*them*, and if no previous entity can be found, it defaults to *it*. Organizations are always *they*. For the purpose of this study, stories that came from the same events (story pairs across both conditions) were filled with the same entities.

Once the sentences from both experiments were complete, I conducted a human participant study on Amazon Mechanical Turk using TurkPrime [91]. Each participant was presented a single story and given the list of 5-point Likert scale questions, validated by

Purdy et al.[92] and used in my previous human-subject study outlined in CHAPTER 4. I exclude categories assessing the long-term coherence of a story as these categories are designed to evaluate an event-to-event system and not event-to-sentence, which is conditioned to map an event to a single sentence at a time. The statement on genre was changed from “Soap Opera” to “Space Opera” Participants were also asked to provide a summary of the story and which of the attributes from the Likert questions thought to be most important for stories. Thus, the list of Likert questions became:

1. This story exhibits CORRECT GRAMMAR.
2. This story AVOIDS REPETITION.
3. This story uses INTERESTING LANGUAGE.
4. This story is of HIGH QUALITY.
5. This story REMINDS ME OF A SPACE OPERA.
6. This story is ENJOYABLE.

If the participants' English was not deemed fluent enough in the open-ended questions, their data was discarded. This left me with 64 in the ensemble and 58 in the baseline condition.

**Table 7 – Test set perplexity, BLEU, & ROUGE (F1) scores, with average sentence lengths for event-to-sentence models.**

| Model | Perplexity | BLEU | ROUGE | Length |
|---|---|---|---|---|
| RetEdit | 71.354 | 0.041 | **11.25** | 9.27 |
| Templates | 203.629 | 0.0034 | 6.21 | 5.43 |
| Monte Carlo | 71.385 | 0.0453 | 10.01 | 7.91 |
| FSM | 104.775 | 0.0125 | 1.29 | 10.98 |
| Seq2Seq | 83.410 | 0.040 | 10.66 | 6.59 |
| RetEdit+MC | 72.441 | 0.0468 | 10.97 | 9.41 |
| Templates+MC | 79.295 | 0.0409 | 10.10 | 6.92 |
| Templates+FSM | 79.238 | 0.0296 | 6.36 | 9.09 |
| RetEdit+Templates+MC | 73.637 | 0.0462 | 10.96 | 9.35 |
| Full Ensemble | **70.179** | **0.0481** | 11.18 | 9.22 |

## 5.4 Results and Discussion

Table 7 shows the perplexity, BLEU-4 scores, ROUGE-4 scores, and average sentence length for event-to-sentence on the testing set for each of the models, ensemble, and baseline. Note that some of the models, such as the sentence templates, make no use of gold standard sentences and are thus poorly optimized to use perplexity, BLEU, and ROUGE scores. In addition to running each model in the ensemble individually, I experiment with multiple combinations of the models to assess which combination makes the most effective ensemble. The full ensemble performs better than any of the individual models with regard to perplexity, as it is designed to combine the models such that each of their weaknesses is minimized. The average sentence length metric highlights the differences between the models, with the templates producing the shortest sentences and the finite state machine taking longer to generate sentences due to the constraints it needs to satisfy.

**Table 8 – Utilization percentages for each model combination on both events from the test set and from the full pipeline.**

| | RetEdit | | Templates | | Monte Carlo | | FSM | | Seq2Seq | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Test | Pipeline | Test | Pipeline | Test | Pipeline | Test | Pipeline | Test | Pipeline |
| RetEdit+MC | 82.58 | 31.74 | - | - | 9.95 | 48.4 | - | - | 7.46 | 19.86 |
| Templates+MC | - | - | 6.14 | 5.48 | 65.7 | 66.67 | - | - | 28.16 | 27.85 |
| Templates+ FSM | - | - | 6.14 | 5.48 | - | - | 56.77 | 32.65 | 37.09 | 61.87 |
| RetEdit+ Templates+MC | 82.58 | 31.74 | 1.49 | 3.88 | 9.1 | 45.21 | - | - | 6.82 | 19.18 |
| Full Ensemble | 94.91 | 55.71 | 0.22 | 0.91 | 4.29 | 41.10 | 0.15 | 0.68 | 0.43 | 1.60 |

I also noted how often each model in the ensemble is used, shown in Table 8, when generating sentences from the eventified testing corpus or from the event-to-event model within the pipeline, across different combinations of ensembles. Utilization percentages show how often each model was picked in the respective ensembles based on the corresponding confidence score thresholds. RedEdit was heavily used on the test set, likely due the train and test sets having a similar distribution of data. On the pipeline events, RetEdit is used much less—events generated by event-to-event are often very different from those in the training set. A majority of the events that fall through RetEdit are caught by the Monte Carlo beam search, irrespective of the fact that RetEdit—and sentence templates—are most likely to honor the event tokens. This is partially due to the fact that satisfying the constraint of maintaining the events tokens makes it difficult for these models to meet the required threshold. The small portion of remaining events are transformed using the templates and the finite state machine.

**Table 9 – Event-to-sentence examples for each model. Ø represents an empty parameter; PRP is a pronoun.**

| Input Event | RetEdit | Tempates | Monte Carlo | FSM | Gold Standard |
|---|---|---|---|---|---|
| ⟨PRP, act-114-1-1, to, Ø, event.n.01⟩ | PRP and PERSON0 move to the event.n.01 of the natural_object.n.01. | PRP act-114-1-1 to event.n.01. | PRP moves to the nearest natural_object.n.01. | physical_entity.n.01 move back to the phenomenon.n.01 of the craft.n.02… | PRP move to the event.n.01. |
| ⟨PERSON2, send-11.1, through, PERSON6, LOCATION1⟩ | PERSON2 sends PERSON6 through the LOCATION1. | The PERSON2 send-11.1 the PERSON6 through LOCATION1. | PERSON2 passes this undercover in the body_part.n.01 and collapses. | Ø | In activity.n.01 to avoid PRP out.n.01 PERSON2 would transport PERSON6 through the LOCATION1. |

Table 9 shows examples of generalized sentence outputs of each of the event-to-sentence models, illustrating some of the trends I alluded to in Section 5.2. Retrieve-and-edit focuses on semantics at the expense of sentence quality. The sentence templates produce output that matches the input event but is very formulaic. Monte Carlo generates entertaining and grammatically-correct sentences but occasionally loses the semantics of the input event. The finite state machine attempts to achieve a balance between semantics and generating entertaining output, however it sometimes fails to produce an output given the constraints of the state machine itself. All of these can be compared to the original next event from the testing set. I also provide examples of the entire pipeline in Table 10, which demonstrates my ensemble's ability to work with an existing plot generator.

**Table 10 – End-to-end pipeline examples on previously-unseen input data. The Event-to-Sentence model used is the full ensemble. Sentences are generated using both the extracted and generated events. Generated events are from a policy gradient DRL model (§4.2) with the "discover-84" verb.**

| Input Sentence | Extracted Event | Generated Events (Event-to-Event) | Generated Sentences (Event-to-Sentence) | Slot-Filled Sentences |
|---|---|---|---|---|
| On Tatooine, Jabba the Hutt inspects the drone barge recently delivered to him. | ⟨ORG0, assessment-34.1, Ø, vessel.n.02, Ø⟩ | ⟨PERSON1, settle-36.1.2, Ø, indicator.n.03, indicator.n.03⟩; ⟨music.n.01, escape-51.1-1, from, Ø, Ø⟩; ⟨PRP, discover-84, to, run-51.3.2, progenitor.n.01⟩ | The ORG0 can not scan the vessel.n.02 of the VESSEL0. PERSON1 decides to be a little person.n.01 at the structure.n.01. the music.n.01 arrives. PRPfinds a lonely person.n.01 on the upper one of the craft.n.02 which is not a personal letter.n.01 but does not respond to hails . | The Jabba the Hutt can not scan the bareboat of the Uss Lakota. O Yani decides to be a little mailer at the airdock. The Music arrives. She finds a lonely mailer on the upper one of the bareboat which is not a love letter but does not respond to hails. |
| Boba Fett has just chased down another bounty, a Rodian art dealer who sold fake works to Gebbu the Hutt. | ⟨PERSON0, chase-51.6, Ø, bounty.n.04, Ø⟩ | ⟨PERSON0, chase-51.6, to magnitude.n.01, Ø⟩; ⟨magnitude.n.01, comprehend-87.2, off, craft.n.02, magnitude.n.01⟩; ⟨PERSON2, amuse-31.1, off, Ø, Ø⟩; ⟨PERSON2, discover-84, off, change_of_integrity.n.01, Ø⟩ | PERSON0 enters the bounty.n.04 and tells PRP. PERSON0 attaches the explosive.a.01 to the person.n.01 who is trying to fix the device.n.01. the magnitude.n.01 doesn't know the craft.n.02 off the craft.n.02. PERSON2 is surprised when PRP learns that the person.n.01 is actually PERSON7. PERSON2 sees the change of integrity.n.01 and tells PRP. | Boba Fett enters the bounty and tells it. Boba Fett attaches the explosive to the peer who is trying to fix the toy. The multiplicity doesn't know the bounty off the bounty. Dark Jedi Lomi Plo is surprised when it learns that the peer is actually Mrs Conners. Dark Jedi Lomi Plo sees the combination off the Orbs and tells them. |

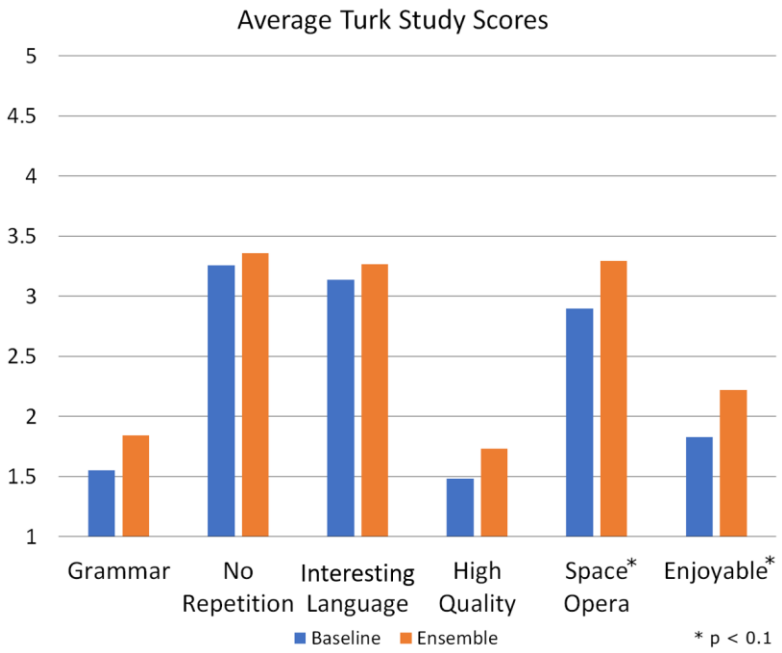


**Figure 5 – Human participant study results, where a higher score is better (scale of 1-5). Confidence values are 00.29 and 0.32, for genre and enjoyability respectively; $\alpha$=0.1. The confidence values for other metrics lie between 0.27-0.35.**

For the human participants study comparing a seq-to-seq baseline to my full ensemble (Figure 5), most metrics were similar in score, which is understandable given that both conditions produced stories that were at times confusing. However, the ensemble consistently outperformed the baseline in terms of quality, maintaining the genre (space opera), grammar, and enjoyablity. Enjoyability and genre were significant at $p < 0.10$ using a two-tailed independent t-test.

## 5.5 Conclusions

Event representations improve the performance of plot generation and allow for planning toward plot points. However, they are unreadable and abstract, needing to be translated into syntactically- and semantically-sound sentences that can both keep the meaning of the original event and be an interesting continuation of the story. I present an ensemble of four event-to-sentence models, in addition to a simple beam search model, that balance between these two problems. Each of the models in the ensemble is calibrated toward different points in the spectrum between the two issues and are thus designed to cover each other's weaknesses. The ensemble is integrated into a full story generation pipeline, demonstrating that my ensemble can work with generated events drawn from a realistic distribution.

Although the system can create longer stories and translate the event representation back into relevant & interesting sentences, the system still lacks casual coherence. In the following chapter, I will transition back to discussing how I improved the *event-to-event* component of the system by integrating causal, rule-based, symbolic methods into my probabilistic, neural model.

# CHAPTER 6. CAUSALLY-CONSISTENT NEURAL STORY GENERATION

Automated Storytelling is a unique area of research since humans are natural "experts", but it is very difficult for artificial intelligence to match this level of expertise. There are several reasons for this: computers are not exposed to all the types of stories we are exposed to, they do not experience the physical world as we do, and they are not capable of connecting ideas or exploring idea spaces in a practical way, among other reasons. However, symbolic systems give generated stories goals and coherence in a way that neural systems (to date) cannot match since neural networks rely on temporal, Bayesian relations between sentences, instead of (narrative) causal relations.

So far, I have shown that even improving upon the state-of-the-art of automated neural story generation with symbolic techniques such as hierarchical event-based generation, policy gradients to guide the network, and an ensemble for event-to-sentence translation, it does not guarantee causally-consistent stories due to the probabilistic nature of neural networks. In this chapter, I will discuss the details of a neurosymbolic system which informs event selection in story generation by using rules to mimic real world physical interactions. I refer to this system as *ASTER-X*, a causal *e**X**tension* to the original *ASTER* pipeline. Examples of the output from ASTER and ASTER-X that have been used in the

experiments for this chapter can be found in Sections A.2.1 and A.2.3 of the Appendix, respectively.

## 6.1 Getting Grounded

When telling a story, one may assume that, just as with the real world, the fictional world does have some rules and conventions, some of which may be explicit, others can be implied. Marie-Laure Ryan named this implication the *principle of minimal departure*, which says that, unless stated otherwise, one is to assume that a fictional world matches my actual world as closely as possible [104]. This means that the fictional world that my agent operates in should have as many similarities to my actual world as I can give it. This poses a problem though; how can the agent acquire models of the explicit and implicit rules of the fictional world? A standard technique in machine learning is to train a model on a corpus of relevant data, such as in the previous chapters of this thesis. While it is possible to learn a model of likely events, recurrent neural networks maintain state as hidden neural network layers, which are limited in the length of their memory and do not explicitly capture the underlying reason why certain events are preceded by others. This is essential because the human co-storyteller may make choices that are very different from sequences in a training corpus—what are referred to as "out of distribution"—and are capable of remembering events and state information for long periods of time. Because of the principle of minimal departure, neural story generation models also fail to capture details that we take for granted in our own lives—details that are too mundane to mention in

stories, such as the affordances of objects. For example, the system would be unable to understand why a cow can be hurt but a cot cannot no matter how much weight you put on it. These models lack *grounding* in the real world.

Symbolic/planning story generation systems achieve grounding by manipulating symbols that refer to concepts in our world. Then, oftentimes, these systems would create causal links between plot points. This is referred to as causal reasoning. If you think of a plot point as a state of the story, certain events need to have happened in order to enter this state and other facts become true as a result of this plot point. For example, if we look at a plot point like "Gemma shoots Paul." Before this event happens, Gemma needs to have acquired a gun. After it occurs, certain *effects* take place—such as "Paul dies" or "Paul loses the game", if it was lasertag. The *pre-conditions* tell the system what needs to occur in order to enter a state, and the *post-conditions*—or effects—occur afterwards. Modelling pre- and post-conditions for ensuring causal coherence has been used in a variety of domains. Magerko et al. made use of pre- and post-conditions for actions so that the individual agents separate from the Game Manager in their game kept the story consistent [105]. Similarly, Clark et al. [106] broke VerbNet semantics into pre- and post-conditions. Pre- and post-conditions are also used in other commonsense reasoning work. Goel et al. [107] used them for design and problem solving, while Event2Mind [108] used pre- and post-conditions for modeling commonsense inference. Mostafazadeh et al. [109] curated a large dataset of cause and effect for story understanding called GLUCOSE, which they then used to finetune transformer models to perform commonsense reasoning. Similarly,

Kwon et al. [110] crowdsourced a dataset of pre-conditions for events in news called PeKo, which was used for pre-condition identification and generation.

In story generation, pre- and post-conditions are used in symbolic story planning systems [22], [35], [44], [111] since it allows for more causally-coherent stories, but with the aforementioned (§2.1) limitations of closed-world story generation. I created a method for acquiring models of the fictional world by blending commonsense, overarching rules about the real world with automated methods that can extract relevant genre information from stories. My agent works as follows. It first generates some possible next events (the 5-tuple variation seen in CHAPTER 5) using a variation of the seq2seq model. Candidate events that are not possible—as determined from the symbolic rules—are pruned from the list of options. A single event is selected and is used to update the agent's belief about the state of the fictional world. This loops for the length of the story and then all the events are converted back into natural language so that people can read what the agent did.

## 6.2 The Agent's Model of the Fictional World

In this section I will describe the two-part model of the fictional world that the agent has access to in order to select actions and update its understanding of the current state of the game world. This model takes the place of the event-to-event component in the full pipeline (Figure 2 & updated in Figure 10).

As shown in Figure 6, the neurosymbolic event-to-event model starts with a neural network such as the ones in CHAPTER 3—a language model that can capture the genre of the dataset you want to emulate, which I refer to as the Genre Model. The output of the Genre Model is fed into the Rule Engine, a series of symbolic processing. The Rule Engine first passes the candidate events into an Event Validator which collects pre-conditions from my version of VerbNet and compares them against the current state of the story. If an event is accepted, the Rule Engine collects the post-conditions from VerbNet and passes them into the State Updater, which updates the story world state. The story state stores a collection of low-level semantic predicates about every entity mentioned in the story so far which a human might infer by reading the story. For the rest of Section 6.2, I will go into detail on what each of these sub-components entails.

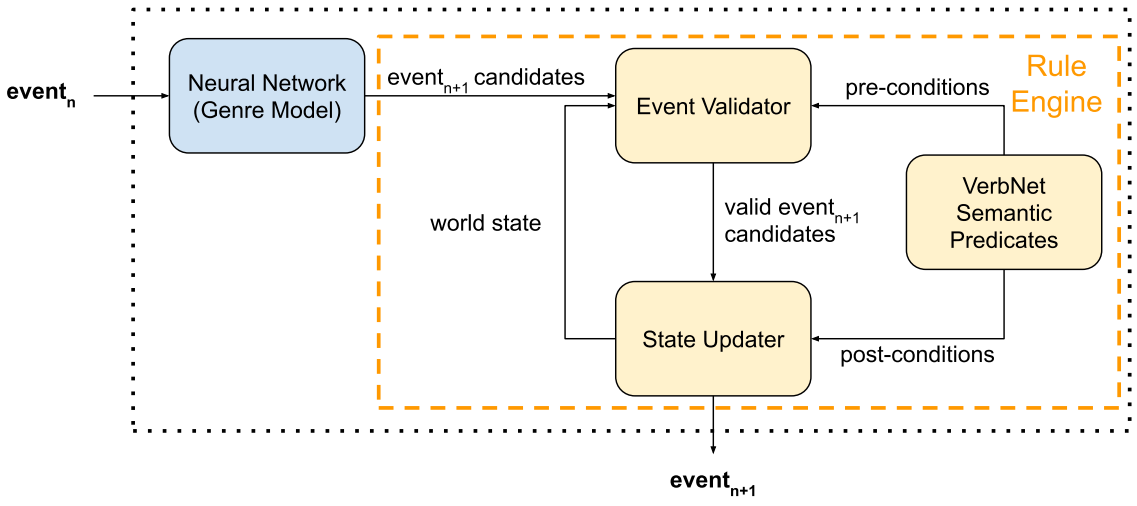


**Figure 6 – Diagram to show the details of the causal Event-to-Event component ASTER. The entire dotted box takes the place of the original Event-to-Event of the full pipeline shown in Figure 2.**

### *6.2.1 The Genre Expectation Model*

Given a corpus of stories from a genre related to the fictional world that the agent will inhabit, I train a genre expectation model. This model provides the probability, according to the genre-specific corpus, that certain events will happen after other events. Specifically, I model genre expectations as a sequence-to-sequence LSTM [70], a type of RNN. I first convert sentences to events. An event is a tuple ⟨ *s*, *v*, *o*, *p*, *m* ⟩ where *v* is the verb of the sentence, *s* is the subject of the verb, *o* is the direct object of the verb, *m* is the noun of a propositional phrase (or indirect object, causal complement, or any other significant word), and *p* is the preposition. As shown in CHAPTER 3, I have found that event representations assist with the accuracy of models trained on stories. In this chapter and in subsequent work, I add the *p* element, which Pichotta and Mooney [75] also found to be helpful for event representations. The preposition is not stemmed nor generalized since there are much fewer prepositions in English than there are verbs or nouns.

Adding the preposition to the event enables me to make closer comparisons between possible syntactic constructions within a VerbNet class since I would not have access to the original sentence's syntax. The story corpus for the genre is pre-processed such that each clause is transformed into an event, which I used to train my Event-to-Event model. This becomes the agent's genre expectation model, giving the agent a pool of likely actions to choose from during its turn. However, it does not guarantee that they are valid or logical actions though, nor would it be able to keep track of long-term dependencies.

Neural networks are particularly good at capturing the style of text even if they do not "understand" it. Therefore, I use a neural network as a *genre model*; that is, a network that dictates what types of events will be seen in the generated stories. In my case, the genre is science fiction. The following language models were trained & finetuned, respectively, on my science fiction corpus [112]. For the event-representation–based version of the genre model, I extracted syntax-specific event tuples based on VerbNet. Although significantly slower to extract, the events are guaranteed to have a VerbNet frame that matches the sentence's syntax.

I previously alleviated some of the problems with sequence-to-sequence (seq2seq) networks [70] by extracting event tuples [58]. However, seq2seq was originally created for machine translation tasks. While this initially seems like it could easily be adapted to other sequential tasks, I realized that even with event tuples they would still lose coherence fast over time when used for story generation. I realized this is because machine translation pairs are independent of each other.

To overcome this difference in the tasks, I altered the seq2seq network to have a persistent hidden state for the entire story. This way, the network can develop a "sense" of what an entire story should look like and hold onto the thread of the story a little longer. Specifically, I altered a seq2seq model with attention and a shared embedding across the encoder & decoder to maintain its hidden state $h$ and its context $c$ across multiple input-output pairs until the story is finished.

In a standard seq2seq network, the encoder equation is as follows:

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right) \qquad [70] \qquad (13)$$

where $h_0$ (the beginning of the encoded input sequence) is ordinarily reinitialized at every encoder-decoder (input-output) pair. However, with the new model, which I call *Seq2Seq2Seq* to denote the memory maintained across sequences, the last hidden state to leave the encoder is passed to the next input encoding (see Figure 7).

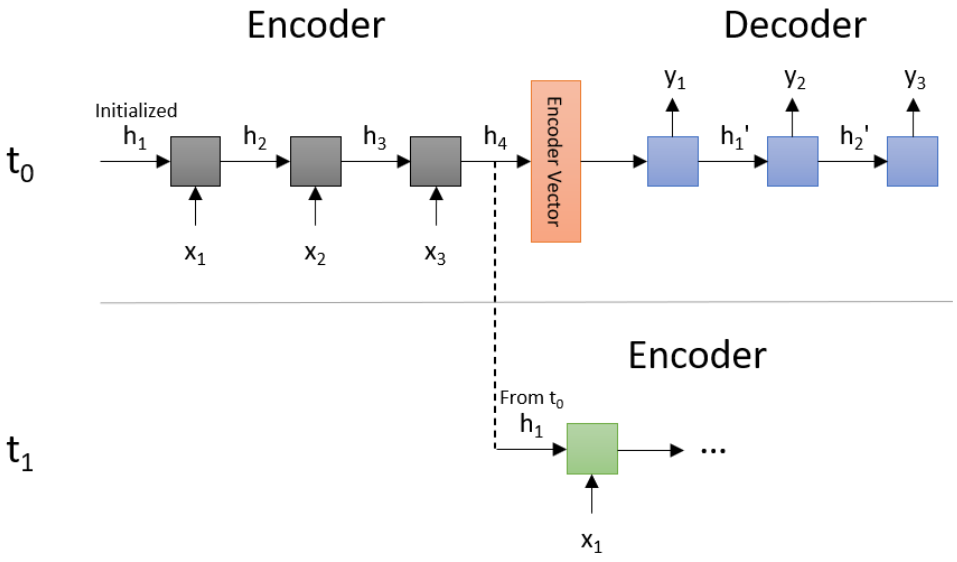


**Figure 7 – A diagram of the Seq2Seq2Seq model bringing the last hidden state of the previous timestep's encoder to initialize the current timestep's encoder.**

While training these networks, I noticed that they train similarly to reinforcement learners in terms of their loss—a more "jagged" pattern emerges instead of a smooth curve. However, I found through pilot studies with humans that these networks perform better

than regular seq2seq networks. For the following event-based experiments I use top-p—or nucleus—sampling [113] to generate the events.

### *6.2.2 The Commonsense Rule Engine*

To help the agent with selecting appropriate events/actions, I acquire a second model of general, commonsense rules about the real world. The purpose of this model is to (a) maintain the story's state of the world and (b) prune out candidate events that would not work for the current state of the story. This is the component that guides the system to create more coherent stories than it would if it was just the language model.

Before I explain my methods in more detail, I first need to define some terms. VerbNet provides a handful of useful information:

- *Syntactic frames.* The syntactic structure that is needed in order to make a sentence with this verb. It uses a constituency-based[6] format (e.g. NP V PP).
- *Selectional restrictions.* The properties that entities need to have in order to fill a role (e.g. +animate or +concrete). A *role* is the name of an argument in a predicate—a

[6] https://en.wikipedia.org/wiki/Parse_tree\#Constituency-based_parse_trees

placeholder. Selectional restrictions provide information about what type of entity can or cannot be put in that role.

- *Semantic predicates.* Predicate logic that denotes low-level semantic information. It has a name and an ordered set of arguments. For example: HAS_LOCATION(Theme, Initial_Location), where the name is "HAS_LOCATION", and "Theme" & "Initial_Location" are arguments. They are read in order with the first argument usually being the subject (e.g. "Theme HAS_LOCATION Intital_Location"). In my augmented version of VerbNet, predicates can have 0-3 arguments.

In VerbNet, each verb class has a set of frames. Each frame is determined by a grammatical construction that this verb can be found in. Within a frame, the syntax is listed, along with a set of semantics. The semantics specify what roles/entities are doing what in the form of predicates. For example, VerbNet would tell us that the sentence "Lily screwed the handle to the drawer with the screwdriver" yields the following predicates:

- ¬ATTACHED(e1, Patient_i, Patient_j)
- DO(e2, Agent)
- UTILIZE(e2, Agent, Instrument)
- ATTACHED(e3, Patient_i, Patient_j)
- CAUSE(e2, e3)

where Lily is the Agent, the handle is Patient_i, the drawer is Patient_j, and screwdriver is the Instrument. In other words: The handle and the drawer were not attached at the beginning of the event (e1), but later become attached (e3). Lily was the one who screwed them together, and the use of the tool was what caused the attachment at the end.

However, the predicates currently provided by VerbNet frames still need to be processed further for my purposes. Preconditions are conditions that must be true in the world prior to the verb frame being enacted. Post-conditions—or effects—are facts about the world that hold after a verb frame has finished being enacted. This information would not be learned by a neural network since it is not explicitly mentioned in stories.

For each candidate event, pre-conditions are by extracting the selectional restrictions for the event's VerbNet class and taking the first semantic predicates of the VerbNet frame. I created a list of selectional restrictions that are mutually exclusive to each other (e.g. +concrete can't be +abstract) and used this to determine whether the selectional restrictions for an existing entity in the state can be in the role it's in in the candidate event. For the semantic predicates, I created categories of predicates: those where an entity can have only one (e.g. has_location), those where it can only be found in one entity's facts (e.g. has_possession—making the assumption that only one entity can have possession of an object), and those that are transitive (together—if Rachel is together with Emma, then Emma is also together with Rachel). I use these categories to determine whether or not the preconditions are satisfied. Any event candidates that are not validated by these two methods are filtered out, meaning they are inconsistent with the current state of the world. Once an action is selected, I use the post-conditions (the rest of the selectional restrictions) to update the agent's belief state about the world. The entire Rule Engine processing loop can be seen in pseudocode in Figure 8 and Figure 9.

**Algorithm 1:** Neurosymbolic story generation loop

**Input:** $LM$, $seed$, $n$, $L$
**Output:** $story$

```
s = new StoryState()
memory = new Memory()
story = []

valid, s = validate(seed, s)
if valid == False then exit()
memory.update(seed)
story += seed

for i = 0 to L do
    E = LM.sample(n)
    validated = []
    for each e in E do
        e.checkStructure() /* check event format if applicable */
        e.fillPronouns(memory)
        if e in story then
            continue                          /* it is a duplicate */
        valid, s = validate(e, s)
        if valid == True then
            validated += e
    ve = validated.select()                    /* currently random */
    memory.update(ve)
    story += ve
```

**Figure 8 – Pseudocode of the neuro-symbolic validation loop, where a language model, start seed event, number of sentence to generate *L*, and number of events *n* to retrieve from the language model.**

**Algorithm 2:** validate

```
   Input: event, s
   Output: Boolean, s
 1 frame = findVerbNetSyntaxFrame(event)
 2 if frame != null then
 3     preds = getPreds(frame)
 4     sorted = getPrePostConditions(preds)
 5     selRestrs = checkSelectionalRestrictions(s, frame)
 6     if selRestrs == null then
 7         return False, null
 8     else
 9         filledPreds = fillPredicates(sorted, event)
10         valid = checkPreds(s, sorted)
11         if valid == True then
12             s.update(selRestrs, sorted)
13             return True, s
14         else
15             return False, null
16 else
17     return False, null
```

**Figure 9 – The *validate* algorithm that the main loop calls. It takes in an event & state and returns a Boolean that says whether or not the event is valid and updates the state if it is. It collects the predicates to turn into pre- and post-conditions, checks to see if the pre-conditions and selections restrictions are valid, and updates the state with the selectional restrictions and pre- & post-conditions if they are.**

Based on the *principle of minimal departure* [104], my agent assumes that when an event occurs, the frame's predicates hold, acting as the agent's knowledge about the actual world. This is reasonable because the frame semantics are relatively low-level and can occur in a variety of scenarios. Whereas the state of the Genre Model is latent, I can use

the facts generated by applying commonsense rules to maintain explicit beliefs about the world that persist until new facts replace them. That is, the drawer and handle will remain attached until such time that another verb class indicates that they are no longer attached. This is important because the agent's belief statement will not be tied to a limited, probabilistic window of history maintained by the genre expectation model. Instead, the state of the world is stored in a dictionary maintaining facts about each entity mentioned throughout the story. The state holds all of the currently-true predicates about the state of the story world.

### 6.3 After Event Generation

Once the new events are generated, I passed them through the rest of my pipeline, which includes the slot filler and the Event-to-Sentence component, respectively. Note that these two components' order is swapped, compared to previous experiments. This is to accommodate BERT [114] (which Devlin et al. released during the time I ran my experiments on the event-to-sentence ensemble [112], [115]) and has shown from pilot studies to perform about the same as the ensemble with a fraction of the computational cost.

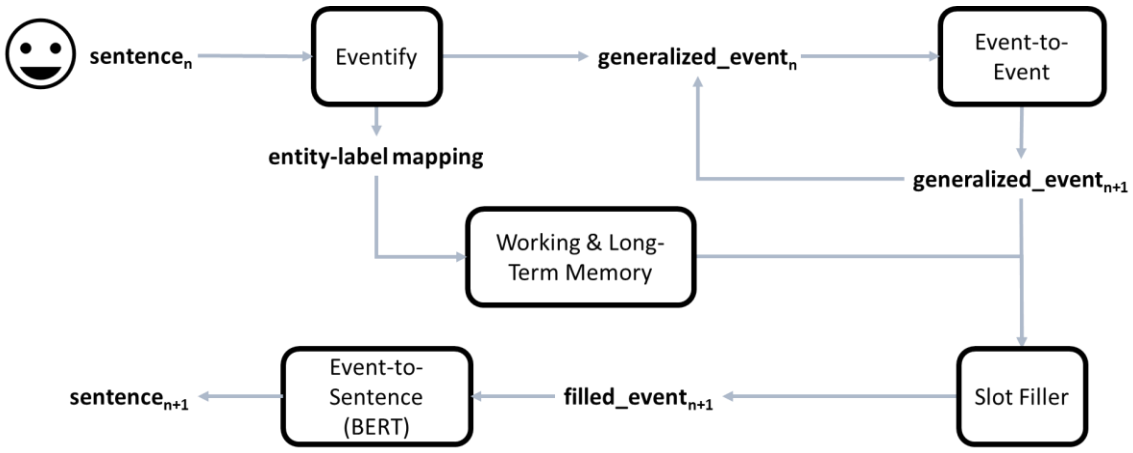


**Figure 10 – The updated ASTER pipeline to reflect BERT usage. Note that the slot filler and the Event-to-Sentence modules swap order.**

### *6.3.1 Memory*

This component, introduced in Section 3.7, of the pipeline does not do any real processing but stores useful information. The memory graph is a simple graph that keeps track of what entities are mentioned during what events, kept in sequential order (see Figure 3). This is the same memory graph initially introduced in my 2016 ICIDS publication [116] and used in my 2017 NeurIPS workshop paper that discusses the full pipeline [117] and in my improved event-to-sentence work [112], [115]. The graph is built over time, tying entities to the events over time. This graph is used to fill pronoun tags (<PRP>) with the appropriate pronouns (they, it, she, etc.). For clarity, the pronouns in these following experiments were, instead, filled with the entity names themselves.

In addition to the graph, the memory module maintains a mapping of what generalized tags refer to what specific entities, thus helping in the slot-filling process. If an entity has

been mentioned before in the story, it is just filled from the memory and doesn't need to be slot-filled from scratch.

### *6.3.2 Semantic Slot-Filling*

After checking with the memory, the first component the event goes through is the semantic slot-filler. Unlike previous iterations of the system, where generalized words were filled in randomly [117], I use a number of techniques to make the slot filling seem more natural.

In my system, the goal of *slot filling* is to fill the parameters in generalized events. As explained in Section 3.2, the nouns have been generalized by looking the words up in WordNet [78]—turning them into *Synsets*—and then taking the semantic ancestor (*hypernym*) 1-2 levels up, and by doing named-entity recognition, giving the entities tags (such as <PERSON> or <ORGANIZATION>) and numbering them as the story goes on. The verbs are looked up in VerbNet, with sense disambiguation resolved by the syntax of the original sentence it was "eventified" from. The reason why the events are generalized is twofold: 1) so that there would be more overlap in the data—which improved the quality of the seq2seq models in CHAPTER 3—and 2) so that the final stories created by the events can be adapted to different topics. Now that the events are generalized, the new, generated events need to be "ungeneralized"—or slot filled—so that they can be turned into English sentences.

However, reversing the generalization process is non-trivial since a single word can only have a single parent hypernym but multiple descendants (*hyponyms*). For example, an event might start with the word "fork", and by eventifying it, it becomes "cutlery". Now the model is trained on events that contain "cutlery", but this can be slot-filled with many options: spoon, fork, knife, etc. Furthermore, each VerbNet class contains many members to choose from (e.g. CARRY-11.4 contains the members "carry", "drag", "hoist", etc.).

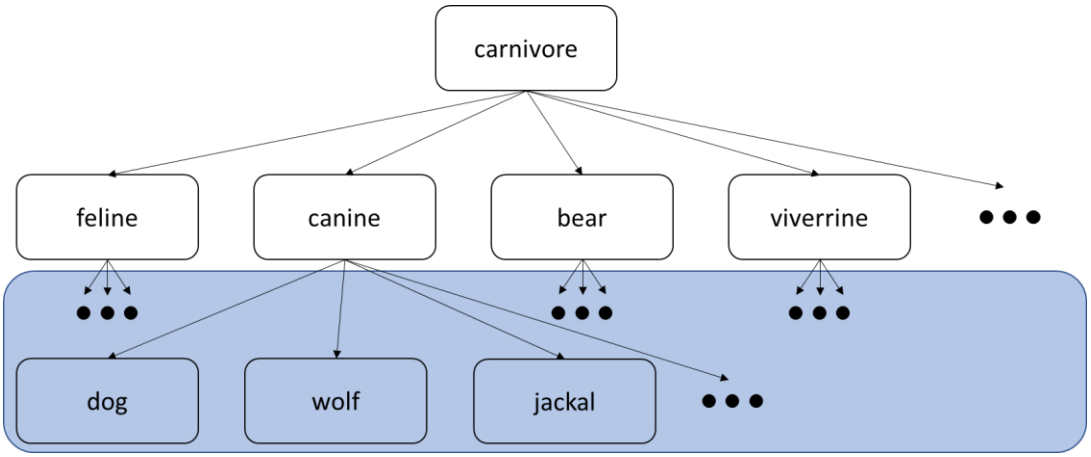


**Figure 11 –An illustration of hyponyms in WordNet that branch from the Synset "carnivore". The blue oval highlights the level of candidate nouns that would be used to fill in for carnivore.**

Prior to this work, I used a random fill-in method that would randomly pick a hyponym that is two levels down in the Synset hierarchy (as seen in Figure 11). The system would also randomly select a verb from the VerbNet class's list of members. The issue with this method is that the hyponyms can be very different from each other. Looking at the diagram, different animals are present in the list of candidate nouns. Also, since each event might have more than one noun or verb to slot fill, there may be a pairing from their respective

hyponym candidates that logically work well together. Let's say that "body_part"[7] was in the same event as the "carnivore" Synset (e.g. ⟨*Synset('carnivore.n.01'), eat-39.1, Synset('body_part.n.01'), EmptyParameter, EmptyParameter*⟩—following the same ⟨*s*, *v*, o, *p*, *m*⟩ event notation), out of both of those words' grandchildren hyponyms, "dog" and "bone" might be seen the most often together in the data Word2Vec was trained on. So "carnivore" would become "dog" and the "body_part" would be "bone", resulting in the filled event: ⟨*dog*, *eat*, *bone*, *EmptyParameter*, *EmptyParameter*⟩.

To improve on the random slot-filling method, I added the following adjustments: 1) added Word2Vec [53] embeddings to find the most semantically-similar hyponyms across an event, 2) reweighted the salience of a word given a history of events, and 3) removed transitive verb options when the event had no direct object parameter.

#### 6.3.2.1 Semantic Embeddings

Since word vectors naturally encode the similarities between words, they were used to determine which noun and verb candidates to fill in. I used Gensim[8] Word2Vec [53]

[7] I removed the Synset syntax for simplicity. All of the common (not named entity) nouns are Synsets.

[8] https://radimrehurek.com/gensim/index.html

embeddings trained on the science fiction TV show dataset (Section 5.1). By using the WordNet and VerbNet categories, word sense disambiguation was already done during the eventifying process, and this information is maintained through the use of the specific WordNet Synsets and VerbNet classes. Therefore, in these experiments, Word2Vec embeddings are sufficient, as opposed to word embeddings like BERT which are context dependent but are use full sentence syntax.

Once the Semantic Embeddings were trained, they were used as follows. First, the nouns are filled. Noun Synset hyponyms were compared pairwise across the event using cosine similarity. The closest noun pair was selected for the final filled event, and then the system would proceed to fill in the next pair until all the nouns were filled. In my example, I would be comparing “carnivore” noun candidates (“dog”, “wolf”, “cat”, etc.) with the noun candidates for “body_part” (“bone”, “head”, “foot”, etc). In this case, the two closest words, taking one from each candidate pool, are “dog” and “bone”. If there is only one noun hypernym left, then I picked the hyponym from its distribution in the dataset, normalized across the set of possible hyponyms for this noun.

Next, the verbs were filled, but since the nouns were filled first, I can use the filled nouns to compare against candidate members from the VerbNet class. The member that was selected was one that was the closest to any one of the nouns that were filled. Back in my example, I would choose a verb member from the VerbNet class (*eat-39.1*) with the

closest cosine similarity to the filled nouns (dog and bone). Here, it would select “eat” out of the options “drink”, “eat”, and “huff”.

#### 6.3.2.2 Event Context History

For each event, I also use information from past events. A Context History Vector is computed by taking the average of the word vectors filled in the previous *n* events. The context vector comes into play in the slot-filling pipeline when choosing nouns. To fill the nouns, I took a weighted average of context vector and the candidate noun. In the final system, I used a context of n=2 previous sentences and a context weight of 0.4. These parameters were chosen by human inspection of stories produced. Notably, when the different sentence context parameters were compared with each other, shown in Table 11, there does not seem to be much difference between the noun and verb changes when the parameters are varied.

#### 6.3.2.3 Exclusion of Transitive Verbs

The last improvement to the slot-filling was removing the transitive verb members in the event’s VerbNet class when there is no direct object in the event. I used a list of transitive and intransitive verbs derived from Beth Levin [118]. In the implementation when I am picking the verb member, if there was no direct object in the event, I would remove the purely transitive verbs from the comparison pool.

#### 6.3.2.4 Comparison

To see how much these methods affected the slot-filling output, I measured how often the verbs and nouns were changed for the different methods when compared to the Random Fill. As expected, the Semantic Embeddings changed the verbs (21%) and nouns (41%) the most. The verb uses changes more than the nouns since I fill in the nouns first and then pick the verb based on those noun choices, making the verbs highly dependent on the noun choice. The Transitive Verb Removal and the Context History Vector were then added separately on top of the Semantic Embeddings (see Table 11). The Transitive Verb Removal created more changes than any combination of weightings and history lengths with the Context History Vector.

**Table 11 – Percentages of the amount of change between the Random Fill method and the addition of the Semantic Embeddings by themselves and the Semantic Embeddings with the other processes. *s* is the number of previous sentences averaged together in the Context History Vector, and *w* is the weight the Context History Vector is given.**

| Comparison to Random Fill | Percentage of Nouns Changed (%) | Percentage of Verbs Changed (%) |
|---|---|---|
| **Semantic Embeddings** | 20.71 | 41.00 |
| **+ Transitive Verb Removal** | 22.04 | 48.72 |
| **+ Context History Vector, s:2 w:0.4** | 21.20 | 42.09 |
| **+ Context History Vector, s:1 w:0.4** | 21.10 | 42.09 |
| **+ Context History Vector, s:1 w:0.7** | 21.05 | 41.60 |
| **+ Context History Vector, s:4 w:0.4** | 21.10 | 42.00 |

In the final system, if the Semantic Embeddings could not be used (because it was the first word selected or there is nothing else to compare it to within the event), then the word

was pulled from the distribution (based on each of the candidates' frequency within the corpus).

### *6.3.3 BERT for Event-to-Sentence*

BERT [114] was used to "translate" the filled events (or filled VerbNet frames, in the case of the symbolic model) into English sentences. To avoid confusing BERT with Synset typography, the events were filled first before being passed through BERT. The code loops through various fillings: masks were placed before and after the event and in between each argument in the event. EmptyParameters were also turned into masks. Each of these masks has the potential to be filled with 0-3 words. Thus, my event representation ⟨ *s*, *v*, o, *p*, *m* ⟩ becomes **s***v**o**p***m**, where each asterisk can be 0-3 word masks. For example, an event like ⟨*dog*, *eat*, *bone*, *EmptyParameter*, *EmptyParameter*⟩ can be as simple as "The dog ate the bone" or be a bit more complex, such as "The very hungry dog ate the cow bone quickly." I loop through BERT translations with different numbers of masks between parameters, and the highest BLEU-scoring sentence (the weighted average of BLEU-1 through BLEU-4) is selected. BLEU is used here so that the resulting sentences are ranked based on similarity to the original ⟨ *s*, *v*, o, *p*, *m* ⟩ input. This guarantees a sentence that does not stray far from the event's meaning. Rarely, this would loop and not find a high-scoring sentence; these stories were skipped and not used for evaluation.

## 6.4 Symbolic-Only Model Baseline

To evaluate whether the neurosymbolic model is better than both neural and symbolic systems, a symbolic-only system was created. Since the symbolic-only method doesn't have any neural networks, I needed a way to generate events or pseudo-events. For this, a VerbNet class and then a frame within the class were randomly selected. I pulled Synsets and named entity tags from the distribution of the original generalized corpus to fill the syntactic frame. To mimic the short-term memory of the neural network approaches, I set the model to pull an entity tag from the current story's memory at the same rate as a pre-existing entity is mentioned in the original corpus:

$$P(old\ entity) = 1 - \frac{\sum_s P(new\ entity)}{\#\ stories\ s} \tag{14}$$

where $P(new\ entity) = \frac{\#\ entities\ in\ the\ story}{\#\ total\ entity\ occurrences\ in\ the\ story}$. In my science fiction corpus $P(old\ entity)$ is 56.95%. The random verb selection and corpus-based noun filling are used in place of sampling from the neural genre model, giving the model far less guidance. The generalized filled frame was then filled using the improved slot-filling approach and "translated" into a sentence using BERT like in the other two approaches. Details for this system can be seen in the pseudocode in Figure 12, and examples of its output can be found in Section A.2.2 of the Appendix.

**Algorithm 3:** Symbolic-Only Model

```
   Input: VerbNetFrames, seed, L
   Output: story
 1 s = new StoryState()
 2 memory = new Memory()
 3 ve = null
 4 story = []
 5
 6 valid, s = validate(seed, s)
 7 if valid == False then exit()
 8 memory.update(seed)
 9 story += seed
10
11 for i = 0 to L do
12     while ve == null do
13         frame = VerbNetFrames.randomPick()
14         filled = frame.fill()    /* based on corpus distribution */
15         if filled in story then
16             continue                          /* it is a duplicate */
17         valid, s = validate(filled, s)
18         if valid == True then
19             ve = filled
20     memory.update(ve)
21     story += ve
22     ve = null
```

**Figure 12 – Pseudocode for the symbolic-only system. The validate() method is the same as in Figure 9.**

## 6.5 ASTER-X Evaluation

To evaluate the three story generation systems (the neural, neurosymbolic, and symbolic), I setup a series of human judgment studies on Amazon Mechanical Turk (MTurk), asking the participants to rate a series of statements identical to those in Section 4.4.2, with the exception of using "Space Opera" instead of "Soap Opera" for the genre statement. In addition to using the open-ended responses to remove non-fluent English speakers, I also used them after my statistical analysis to gain insight into why people rated the stories the way they did. Each participant was paid $4, with the survey taking about 20 minutes to complete. After removing non-fluent English speakers, I collected a total of 50 participants.

Unlike in the REINFORCE experiment (CHAPTER 4) where examples were picked across conditions and then translated by humans, these stories were randomly selected and then translated into sentences using my semantic slot filler (§6.3.2) and BERT (§6.3.3). Stories were generated from each starting sentence/event in the testing set and cut off at 6 lines. If a story (in any condition) was unreadable at first glance, I resampled it. Five stories were pseudo-randomly selected from each condition. One story from each condition was randomly selected and shown to the participants in a random order—each participant seeing stories across the three conditions. To match the symbolic model, pronouns that were produced in the events (the neural and neurosymbolic model) were filled with entities using the memory module. The neural-only system here is a Seq2Seq2Seq model with mildly pruned output to get rid of poorly-constructed events (e.g. events with a noun where

a preposition is supposed to be or an EmptyParameter in the subject), but it was **not** pruned for causal consistency.

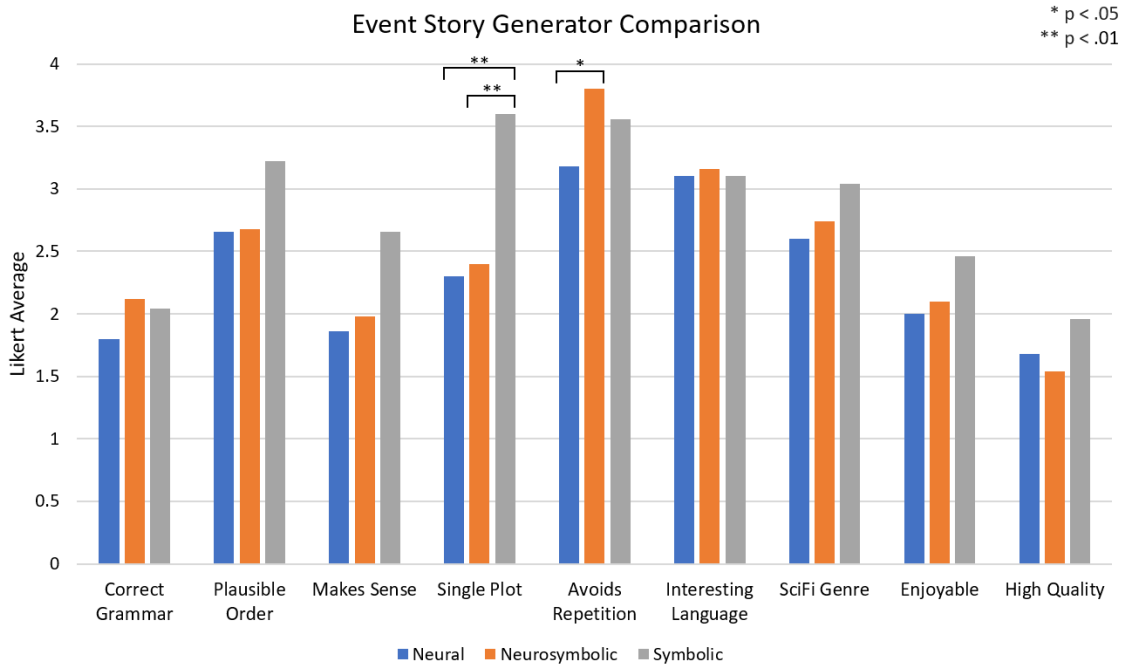


**Figure 13 – A graph of Likert averages comparing the neural-only ASTER model, the neurosymbolic ASTER-X model, and the symbolic-only model across the nine attributes. The symbolic model performed significantly better than the other two models (both $p < 0.01$) in terms of having a single plot. The neurosymbolic model was significantly better than the neural model for avoiding repetition ($p < 0.05$).**

### *6.5.1 Results & Discussion*

The Likert rating averages are reported in Figure 13. The neurosymbolic system performed about the same as the neural-only system across the attributes. However, after the Likert ratings were turned into rankings, a Friedman test showed a significant effect of the model

type on *avoiding repetition* ($p < 0.05$), and a Nemenyi post-hoc comparison test showed that the neurosymbolic system did significantly better in avoiding repetition ($p < 0.05$).

Somewhat surprisingly, despite entities just being pulled from the distribution, the symbolic-only system seemed to perform better than the other two systems across almost all qualities. However, a Friedman test showed a significant effect of the model type on being perceived as a *single plot* ($p < 0.01$), and a Nemenyi post-hoc test showed that the only time the symbolic system did significantly better was where it was more often perceived as a single plot (both $p < 0.01$ compared to neural and neurosymbolic models). There are a couple of reasons why the symbolic system performed better than the neurosymbolic system: (1) the symbolic system used VerbNet syntax frames, which more easily relates to English grammar than my event representation does—maybe this even improved BERT's event-to-sentence capabilities, and/or (2) it could be that pulling from the original generalized corpus's distribution does better than using nucleus sampling with the Seq2Seq2Seq models. This would mean that between what the Seq2Seq2Seq learned and how it was sampled made things *too novel* to make much sense. However, all of the three systems' stories had poor readability, and I believe that this would greatly affect participants' ability to judge the stories. I will look more into this in the next section by using full sentences.

## 6.6 When Life Gives You Transformers, Use Them as your Genre Model

GPT-2 [1] is a large-scale transformer-based language model capable of generating natural language sentences that seem more fluent, consistently out-performing previous approaches like LSTM-based Seq2Seq [119], [120]. In this section, I discuss experiments using a neurosymbolic version of GPT-2, which I refer to as **ASTER-XT** (ASTER-X with Transformers). Since GPT-2 can handle sentence generation better than Seq2Seq and to avoid finetuning on my event tuple representation (so that I can leverage the power of GPT-2), the story generation pipeline is slightly modified. Instead of event tuples, I used a finetuned version of GPT-2 trained on full sentences from the science fiction dataset (§5.1) with named entities generalized & numbered. Sentences were sampled from the finetuned GPT-2 using a standard beam search. Note that this system does not use WordNet Synsets. Instead, I relied on GPT-2's ability to "remember" some context from previous sentences, especially since common nouns are far easier to find relationships between than named entities (e.g. "llama" and "grass" are more easily connected than "llama" and "Sam" since the former would probably occur in more stories together).

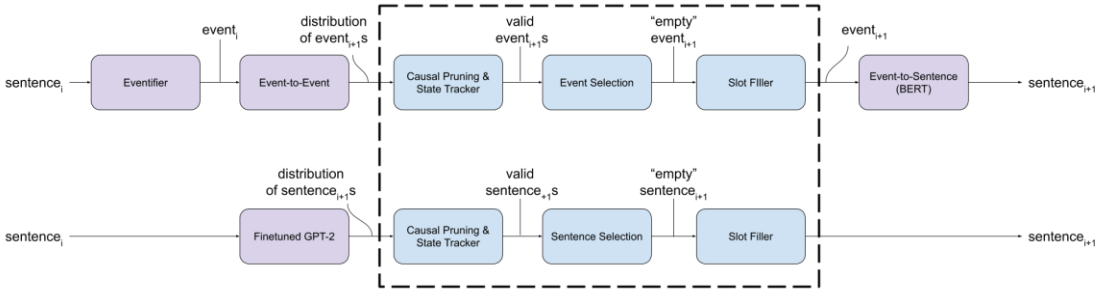


**Figure 14 – A comparison between the event tuple-based (top) and the sentence-based (bottom) neurosymbolic pipelines. The pipelines have been flattened for comparison.**

The eventification process and the event-to-sentence component were not needed when there are already full sentences. To enable the symbolic component to work with GPT-2's sentences, I ran a dependency parser to extract the verb, the nouns, and any prepositions—similar to the eventification process. With the words extracted from the sentence, the verb would be stemmed and matched to a VerbNet class, and the closest syntactic frame would be filled with the extracted words. Once this was in place, the Rule Engine was used unaltered; the symbolic reasoning, event selection, and the filling of named entities all remained the same. See Figure 14 for the comparison between the event tuple-based and the sentence (GPT)-based pipelines. Example output stories of the Finetuned GPT-2 and ASTER-XT can be found in Sections A.2.4 and A.2.5 of the Appendix.

## 6.7 ASTER-XT Evaluation

I conducted two experiments with ASTER-XT. One where just the neurosymbolic and neural models are compared; another with neurosymbolic, neural, and symbolic models. Using the same exact methods as in the previous section (§6.5), I began with an experiment just comparing the neurosymbolic system to the neural-only system, which is just the finetuned GPT-2. I collected 50 data from participants. Compared with the previous experiment, each participant would be reading one less story since there is no symbolic-only condition, and the sentences were far easier to read since they were more fluent in terms of grammar and less terse.

However, because of the event tuple-based results with the symbolic system outperforming the neurosymbolic system in the first, ASTER-X experiment, I decided to run a second GPT-2 experiment comparing all three: neural, neurosymbolic, and symbolic story generators. Because the symbolic system does not use a neural network at all, it is the same exact system (§6.4) as was used in the previous, event-based experiment. I recruited another 50 participants that were different from the other GPT-2 experiment above.

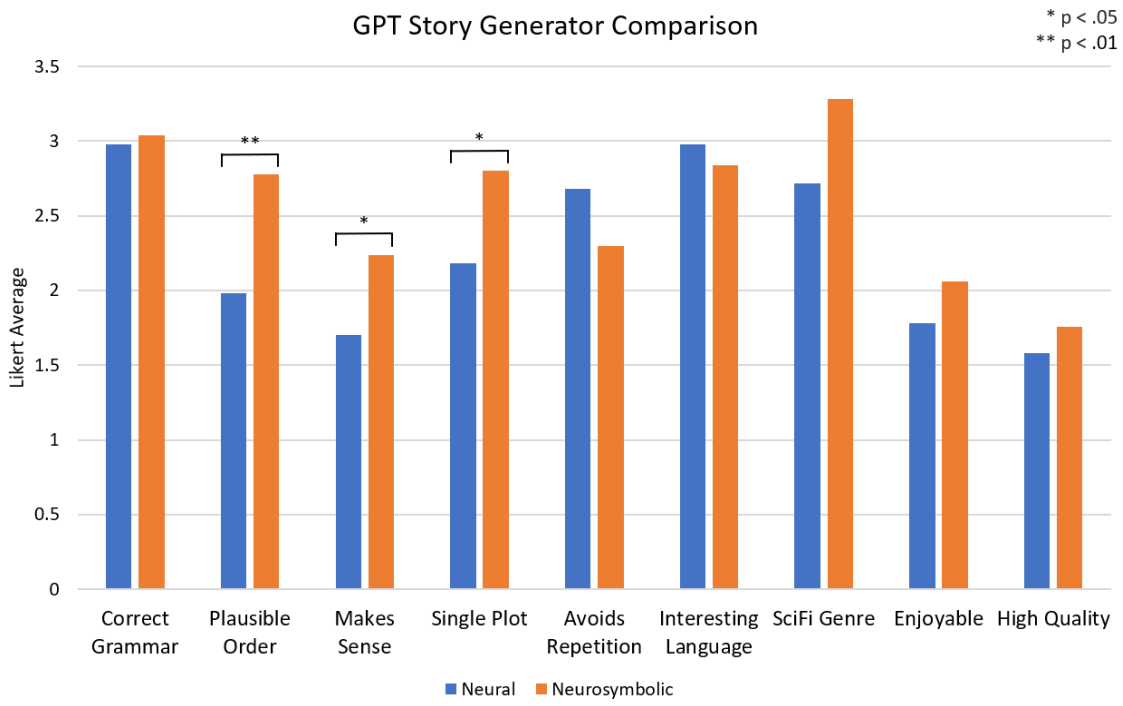


**Figure 15 – A graph of Likert averages comparing the neural-only model and the neurosymbolic model across the nine attributes. The neurosymbolic model was found to be better than the neural only model in terms of plausible ordering ($p < 0.01$), having the lines make more sense given the ones before and after them ($p < 0.05$), and having a single plot ($p < 0.05$).**

### *6.7.1 Results & Discussion*

Figure 15 shows the average Likert scores over the nine questions, comparing the neurosymbolic model to the neural-only model. The neurosymbolic model did at least slightly better in all categories except for avoiding repetition and having interesting language. Based on open-ended responses from the participants, this is mostly because the neurosymbolic system repeats a lot of the same characters, and because I decided not to use pronouns in these experiments, character names became cumbersome for the

participants to read. Likert ratings were turned into rankings, and a Friedman test showed a significant effect of the model type on perceived *plausible ordering* ($p < 0.01$), *making sense* ($p < 0.05$, and having a *single plot* ($p < 0.05$), with the neurosymbolic system performing better. These results are highly promising, considering how ubiquitous GPT has become in the field of natural language processing.

The results of the second GPT experiment are shown in Figure 16. Again, Likert ratings were turned into rankings, and a Friedman test showed a significant effect of the model type on perceived *grammar* ($p < 0.01$), *repetition avoidance* ($p < 0.01$), adherence to being in the *genre of science fiction* ($p < 0.01$), *enjoyability* ($p < 0.01$), and *quality* ($p < 0.05$). For the most part, the neurosymbolic system outperformed the symbolic-only system except with avoiding repetition, where a Nemenyi post-hoc comparison test showed that both the neural and the neurosymbolic systems did significantly worse ($p < 0.05$ & $p < 0.01$, respectively). This is most likely due to the entities being pulled from the corpus's distribution instead of being conditioned on history, as neural networks do. The symbolic-only system did slightly better than the neurosymbolic system with seeming like a single plot, but this result was not significant.

The Nemenyi post-hoc test also showed that the symbolic-only model does significantly worse in terms of genre than both the neurosymbolic model ($p < 0.01$) and the neural model ($p < 0.01$), despite being sampled from the same distribution as the corpus. This proves that the genre model is a necessity for maintaining style when compared to a

generic, open-world symbolic model, and that finetuning GPT-2 did help capture the science fiction genre.

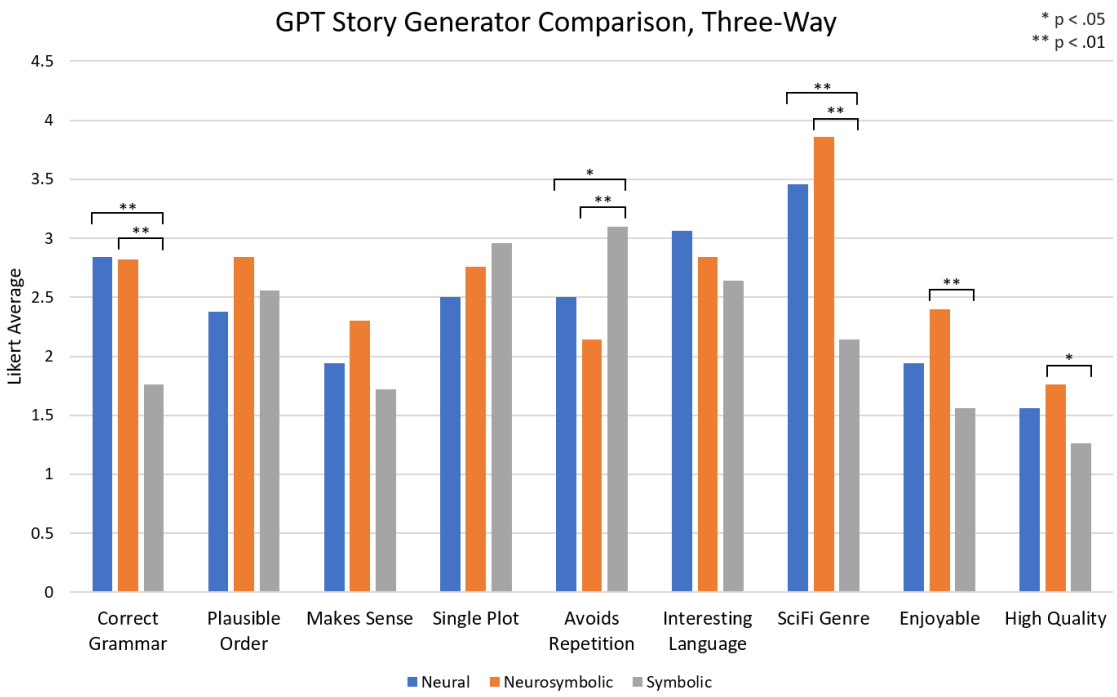


**Figure 16 – The average Likert scores for the three-way comparison using GPT-2. The neurosymbolic GPT model does significantly better than the symbolic-only model in grammar, genre, enjoyability, and quality. The symbolic model performed significantly better with avoiding repetition.**

The neurosymbolic model does significantly better in terms of enjoyability ($p < 0.01$) and quality ($p < 0.05$) than the symbolic-only model, and again, performs about the same but slightly better than the neural-only model, which was seen in the previous experiment. People also reported that the grammar was significantly worse with the symbolic model than the neural model ($p < 0.01$) and the neural symbolic model ($p < 0.01$). This could

imply that grammar greatly affects enjoyability and quality, but more experiments would need to be done to verify this. With GPT-2 as the genre model instead of events, the feeling of the symbolic-only model being significantly better in terms of plot when compared to the neurosymbolic or neural models was drastically muted.

Curious about the correlation between the three questions I have been using to measure various aspects of coherence (the "plausible order", "makes sense", and "single plot" questions), I ran an analysis across all of the experiments discussed in this chapter. I calculated Cronbach's alpha ([121]) to be 0.7997 across the "plausible order", "makes sense", and "single plot" questions, across all three experiments. This shows that there was some correlation across the questions, meaning that they could be calculating the same thing: coherence.

# CHAPTER 7. BROADER IMPLICATIONS

In review, I begin working toward more coherent story generation by separating the problems of story plot generation and making syntactically-sound sentences improves plot generation (CHAPTER 3). I have done this by extracting "events" from the original sentences that abstract away the core semantic meaning such that there is more overlap in the data that the neural network sees while training. It also lent itself to the creation of an entire pipeline for story generation. This was some of the earliest neural story generation work.

I have discovered that by creating a model that gradually heads toward a goal improves the perception of the story having a single plot and having a more plausible order of events (CHAPTER 4). Starting with a base event-generation model, I used policy gradient reinforcement learning to retrain the model. This work pushed the trend of controllable neural networks.

After seeing the compounding error of feeding event-to-event output into the event-to-sentence model, I improved on the basic seq2seq model by creating a cascading ensemble of methods for event-to-sentence (CHAPTER 5). This gave the model freedom to strike a balance between correctly translating the events into sentences that have the same meaning, while also making sentences that are interesting to read.

Finally, I introduce neurosymbolic story generation (CHAPTER 6). I created a commonsense-based symbolic system that used VerbNet to create causal continuity for story generation. The rules determined the validity of candidate events proposed by a neural network. It created stories that were seen to be more consistent than a neural-only model and more enjoyable & genre-appropriate than a symbolic-only model.

In summary, to create a decent story generation system, the agent needs to, at the very least, have an understanding of what **pre-existing stories look like** and basic understanding of **how things work in the world**, and a **way to come up with interesting, new events** based on the first two. These events can be in a variety of formats and should be designed for what works best for your task. Finally, this can be said about any computational subfield, but it is important to realize the strengths and weaknesses of every tool available. This means educating yourself in the history of the field and seeing what has worked (and why) over time, while also keeping track of the "bleeding edge" technology—all as long as time allows, of course.

Any of the systems mentioned in this thesis are far from the ideal (or even a good) story generation system. Storytelling is incredibly complex, and this field is still nascent in its development, despite the many decades working on symbolic systems. There is still a lot to be done, so to help you get started here are some of the next areas I think would help push the state-of-the-art in story generation.

## 7.1 Possible Future Directions

**"Real" Reinforcement Learning.** I proposed [122] a reinforcement learning technique based on Deep Q Networks that can learn to use these models to interact & tell stories with humans. In traditional reinforcement learning (RL), RL agents learn the policy incrementally by trial and error, attempting to find correlations between states $s_i$ and future expected reward, $Q(s_i)$. Deep Q networks learn to approximate the expected future reward for states with a neural network. Deep Q network agents have been shown to be able to play complicated games such as Atari video games [123] and text adventures with limited action spaces [124]–[126]. My event representation (or the event representation of your choice) turns an infinite number of actions (any natural language sentence) into a large, but finite action space. Still, we cannot perform exhaustive trial and error learning while training a deep Q network to tell a story.

Using my neurosymbolic agent and pruning out invalid events, I can help the agent to search through possible future states for those that return the highest reward, which will, in turn, help the agent converge faster in situations where the reward (e.g. reaching a particular state) is sparse. Details on the flow of information through the DQN can be seen in Figure 17.

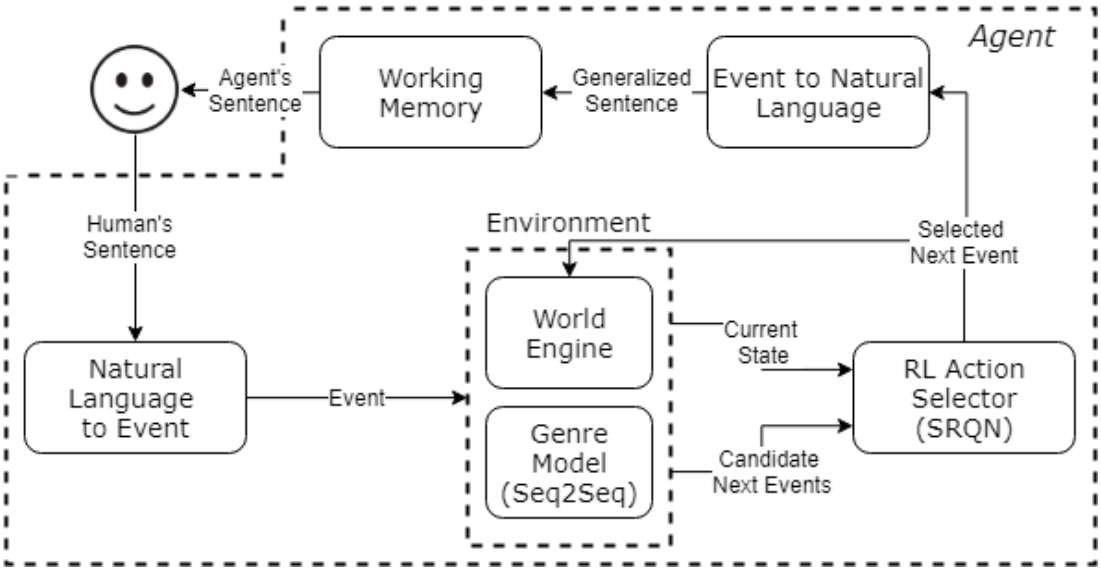


**Figure 17 – The entire proposed pipeline of agent once the reinforcement learner is trained.**

While the DQN is exploring during training, the previous event in the story is passed into the environment, such as my neurosymbolic event-to-event model (§CHAPTER 6), which provides possible actions (events). The agent chooses to exploit its policy model or explore randomly, and once a valid event is picked, the story state is updated. Because there is a rule model, I can conduct multi-step lookahead as well, wherein the agent explores several steps into the future before using the reward to update the policy. Each event that is picked should bring the agent closer to its goal for the story, which is a pre-defined event.

**Character-based Generation.** All of the work in my thesis focused on generic plot progressions, but many human-written stories focus on characters and their journeys &

relationships. Character stories are important because we can relate to characters that are like ourselves, or we can suspend disbelief and see the world through someone else's eyes. Making a system like this involves character modeling [71], [127], relationship modeling [128], [129], and affect modeling [130]. By being able to model how people act and interact with each other, we can create richer, more engaging stories.

**Dialog and Meta-Understanding.** Related to character-based story generation, is dialog understanding and generation. Dialog is something I intentionally avoided during my PhD career due to the complexity it adds. Dialog adds a layer of occlusion to our understanding of the story. For example, if someone says they did something, did they actually do it or did they just say that they did (grounding)? Were they being sarcastic or sincere when they said it (intention)? Who did they say it to (audience)? We need ways of modeling all of these systems. Because of its complexity, it was not something I could jump right into, but systems like the ones I made can be used as building blocks toward more sophisticated systems like this.

**Story Arcs.** It is one thing to learn how stories should be told on a sentence-by-sentence basis, but what if the system could focus on emotion, plan ahead, and lead up to a climax and fall? Like goal-driven narrative, this type of story generation would involve meta-planning. It would require an agent to keep track of the individual sentence-generating agents to make sure, not only are those agents achieving their goals, but that the meta-goal is also being reached.

# APPENDIX A. STORY EXAMPLES

## A.1 Examples from CHAPTER 4. Reaching Major Plot Points

### *A.1.1 DRL*

Sister died.
Banana executed during the visit.
Banana adopted the girl.
The girl looked like the mom.
She was appalled.
Penelope detested the jungle gym.
==========
The woman informed him.
She brings preoccupation.
She regrets.
It devolves.
The appearance recovers it.
It holds.
A part despises it.
==========
The girl looked at the camera.
She verified the change.
She took off.
The boy stayed.
He insisted.
He tried to get that article.
He fancied.
==========
The man tied up the woman.
The man had to improvise.
The woman refused.
Janice Ian was staying by herself.
Janice Ian flaunted to the man.
Janice Ian stayed.
Time had wings.
They appreciated change.
==========

They scurry from death by the enemy.
They train.
The person served a use.
The person does the finger.
The person was blamed for being protective.
The usefulness grew.
The richer buy from Sam.
The calamity’s suffering engaged.

### *A.1.2 Seq2Seq*

She reminds him.
She has doubt.
She annoys Bob.
Alice sues him because of the breach.
He refuses to sing.
He bellows.
He testifies.
Panic will rule.
Make narration preposterous.
He is clueless about the thing.
He is dismayed.
==========
My cousin died.
Alexander insisted on a visit.
Alexander met her.
Barbara commiserated with Alexander.
They hugged.
The gathering dispersed to Hawaii.
The community remembered their trip.
They robbed the pack.
They adored the party.
==========
She was looking at football.
She found a master player.
She was impressed by the header.
John is dehydrated.
He has fatigued his back.
Snake ruined the sector.
The sector turned into a misfortune.
The sector chanted die to prune.

The dining area began with everyone.
John told Kelly.
==========
He was handcuffed to the person.
They encouraged him.
He improvised.
He implicated the townspeople of the remains.
He bargained.
He hid his inheritance.
He consulted his feelings.
He bolted.
A decade passed.
Cheese inherited bulk.
==========
Snake crawled in a circle with caution.
Ramesh hypnotized.
He is ashamed of the world.
The substance destroyed the tree.
The tree fell eventually.
The tree chanted pick to the world.
The dining area began with everyone.
Ramesh told Suresh.
The neighbor proposing is sweet.
The neighbor got money.

### *A.1.3 Gold Standard (Translated)*

She reminded him.
She had a child.
They wished for more time.
She accorded with history.
Let the record say.
He understood.
He fancied a shave.
He shaved because he needed to.
Prepare to halt shaving.
Sam made it to the meeting with Ray.
==========
Our in-laws disappeared.
Looks indicate to the South.

Olaf suggests for a lawyer.
They attempt to gain.
The neighbor called for a ransom.
They declared for an official.
The official deduced for Olaf.
Olaf sustained a tragedy.
Misfortune had him.
==========
She looked at the bomb.
Jenny feels for the child.
Jenny calls out to god.
Jenny exclaims to her platoon.
She seized the prisoner.
The risk is encircling.
The patrol is defeated.
He reunites with Jenny.
He urges her.
She goes on reflecting.

## A.2 Examples from CHAPTER 6. Causally-Consistent Neural Story Generation

### *A.2.1 ASTER*

Gillnitz trained to yellow by Kerane.
Gillnitz is quested.
Gillnitz trickled to vamoose with Kerane.
Davey Crockett Motor Court let bridge in inheritors.
Davey Crockett Motor Court ran.
The Anton was quested him.
==========
Stranger abandoned frozen man.
Frozen accused around USS Saratoga.
Out predicted victory lap in are.
D Thomas Edison revealed welcome news.
USS Saratoga came for inside.
Inside aided USS Saratoga into planet.
==========
Crewmen Muniz took on man.
K the Brown 9 grew.
Brown 9 suffered.

Sheeka urged Aaron Downey.
Thing tasted to induce.
The thing is importuned here.
==========
Millennium Falcon received second gear via first-aid station.
Back attracted Sklar.
Sklar debated to understand back.
Back screened calorimeter.
Stranger snapped to eat.
Absolute value took absolute value on stranger.
==========
President stepped through south east coast.
President owned by Angel Media Group.
Scooti Manista continued fact.
Scooti Manista pained fact.
's a Scooti Manistato surmised.
Scooti and Manistat had anticipated success.

### *A.2.2 Symbolic-Only*

USS Carolina made it.
USS Carolina launched round into numeration system.
Commander took USS Carolina.
Commander whipped to USS Carolina.
USS Carolina sapped Nechani.
Nechani forced USS Carolina storage.
==========
Signal seemed to even.
Signal kept up power through like.
Output ate in life.
Electricity (has knocked all together.
Signal intended Lieutenant Kirk.
Lieutenant Kirk reminded. signal.
==========
Silence came.
Morokko misapprehended silencer.
Morok turned to Sarn silence.
Sarn aimed silencer.
Sarn devoted silence to Morok.
Sarn (amplified drop).
==========

The crew though hard.
Crew longed Ellen Persich.
Aerator said to sentry droid aerator.
Sentry droid exhaled Allen.
Allen sent sentry droid in horn.
Horn withdrew to sentry droid.
==========
Female harmed to charge.
Bat proceeded Jilka Zan Zentis.
There begged bat for Jilka Zan Zentis.
Talkie has sailed to Batang Island.
Bat answered to home.
Home picked up about bat.

### *A.2.3 ASTER-X*

Starling explained interaction on wattle and daub.
Starling has said.
Starling was declared thing winner.
Starling has persevered.
Starling squared to try.
Starling brought life form in Mandelbrot Set.
==========
Gonk Droid stained to thunder.
A wired screened by a wire.
Darmok looked with mining.
Mining stooped at Battle of Demon's Run.
Torchwood Three topped Saroosh.
Saroosh took Planck's radiation law through base.
==========
Flatbush handed out shield.
A the Mlee said no.
Mlee chanced across Flatbush.
Mlee met by Flatbush.
Flatbush consumed its densities.
The Flatbush ordered terminal line.
==========
Felix tried to uplift.
Felix wound Ensign Kane.
Felix toured Ensign Kane.
A Felix Yakeda feat.

Ensign Kane chanced across.
Ensign Kane discarded onto missile.
==========
Grand Nagus cut chamber.
Secretion garlanded to screen by Tor.
The open water - sheltered areas.
Water relearned to rationalize.
The coded guessed correctly.
Code never ran out.

### *A.2.4 Finetuned GPT-2*

After working for Jabba for a few years, Boba had made a name for himself becoming Jabba's favorite bounty hunter.
Shahna jabba's owner, Skere Kaan jabba, had been killed by a mysterious man who had come to collect jabba's stolen things.
Jabba's body had been taken into custody.
Jabba's gun had no effect on jabba's, and Boba was also taken away by Ag Robinson.
Ag Robinson jabba's body was taken to Outbuilding where Ag Robinson was put to die.
Jabba's body was taken to Vis Uban.
Jabba's body was left there by Boba.
Jabba's body was left there by Boba.
==========
They were sent to destroy the voxyn.
Palamar was sent to the Chorley to retrieve it.
Haninum Tyk Rhinann was ordered to do it himself.
Nick Months Ago, Lizzy, and Margaret Blaine are brought in.
They are taken into custody.
Garage is left behind in the Saratoga.
Pacifica is taken away, while Gaashah and Dinua are left behind.
Fhost, Lynch and Jessen are taken to the Midtown Manhattan.
==========
Luke and Ben Skywalker narrowly survive an attack by a Sith assassination squad.
Ben Skywalker and Luke escape from the Mistfall and find a Pacifica where they encounter a strange looking man named Benbeck.
Luke explains that Luke was the man who attacked Ben Skywalker.
Thus, Luke was responsible for killing Ben Skywalker.
Thus, Luke was responsible for killing Ben Skywalker.
Thus, the assassin was the one who caused the murders, and the killer's killer is the man who was responsible for killing Ben Skywalker.

Ben Skywalker explains that Ryymax's body was found on the floor of a Bannerman Road, and Ben Skywalker's body was taken.
Ben Skywalker claims to have killed Ben Skywalker's head with a stick.
==========
Anakin Solo is now eleven -- and headed for the Jedi academy.
Katharsis, Kassia and Griffiths discuss the possibility of the Trail Bike's escape.
Mclaren Natatorium and Neelix Regretfully are convinced that it is not a Dogsled but rather a Dogsled that has been left behind, a Dogsled that is Now eleven
Dead.
However, Anakin Solo thinks the Arias Expedition's story is a lie and that the Submarine has no physical evidence.
Peaceful Slumbers Funeral Parlor, who has never seen any Automobile in his life and never heard of anything like Anakin Solo, believes it is a fake.
However, Anakin Solo's claims that Kiva has been dead for 85 Years Old and that Anakin Solo was killed by Kago Darr Train in an accident while on Peaceful Slumbers Funeral Parlor.
Anakin Solo says that he's been seeing some strange lights and people all night, which makes Kiva suspicious.
==========
A jogging couple finds eight-year-old Teena Simmons sitting dazedly in her front yard.
Imperial Lieutenant Commander Avvan Hocroft and Okala are walking along the Logan International Airport.
Viner asks her to stay and she says she is.
Eight-year-old Teena Simmons, Fourteen Hours Earlier.
Eight-year-old Teena Simmons, 2325.
Viner's father tells her the story of how the boy went into stasis when he was in stasis.
I remember.
Viner was born 500 Million Years into a dark Door 40 on Telford.

### *A.2.5 ASTER-XT*

His contact said that two refugees were being held prisoner by the Galactic Empire.
Salman Kohl said the Nausicaans were trying to take advantage of a situation in the Mgm that they had been working on for years.
However, Thot Pran Of The Breen was not going to allow the refugees to be imprisoned, and that the refugees are being held against their will.
Dala said Thot Pran Of The Breen was trying to save them, but the refugees were being held at gunpoint.
Thot Pran Of The Breen said Thot Pran Of The Breen could not do it.
Gelth had no intention of doing so and could not let Gelth's people go.
Generoo said that Thot Pran Of The Breen had no intention of killing anyone.

Gelth had no intention of destroying the Psa, but had a sense of safety.
==========
The young Padawan Dooku had recently been appointed to Jedi Thame Cerulian.
However, he did not want to be on the list and ordered Messaline to stay on it.
Padawan Dooku had to stay on it, so he could be seen by Padawan Dooku's staff.
Arturo Villar was Arturo Villar Padawan Dooku had been keeping a close eye on.
Arturo Villar was Arturo Villar who was being watched.
Qonos was not in the Accommodation Centre of Arturo Villar.
Padawan Dooku's staff is in the Patahk of the Galleon, waiting for a Buggy to arrive.
Kaas City tells Arturo Villar's that Padawan Dooku's staff have found the body of the dead Cybermite.
==========
Jedi sisters Alema and Numa Rar are attacked, aboard the Nebula Chaser, by the Yuuzhan Vong's newest creation, the voxyn, which can hunt down Jedi through their Force presences.
Yuuzhan Vong and his new The Chaser have been sent on a suicide mission.
Alema's body is taken.
The Chaser is taken, along with Numa Rar.
Yuuzhan Vong's body is discovered, and The Chaser is taken.
Alema's body is taken, and Yuuzhan Vong's body is taken by an mysterious person.
The Haul's weapons are missing, and Jedi's body is taken.
Yuuzhan Vong's body is found lying on the floor of a 2012 New York City.
==========
The Punishing One enters the Thanos system.
Brian Previously is on Brian Previously's way to Sherlock Holmes tube to deliver a report to the Shiar, where Brian Previously is being questioned by Carr.
Phyllis Trupp's body is in a coma, and Dastari is being questioned by Chez Sandrine.
Hallett's body is being kept for the Feeder for medical examination.
Gavin is being held in a holding Broton, and Corey is being held captive.
Corey is taken into custody and Gavin is being tortured.
Corey is being tortured by Trevis's captors and kept for the use of force.
Chez Sandrine is being interrogated by Dee Dee Blasco, who is interrogating Munlali Mafir as Gant is being tortured by the Vellas.
==========
Luke and Ben Skywalker narrowly survive an attack by a Sith assassination squad.
They are soon be beamed aboard and taken away by Ben Skywalker Luke, Gara Petothel.
Ben Skywalker's body lies on the floor.
Luke is taken away.
Luke is taken back to Mekorda.
Luke is taken into custody and Pompeiian is questioned.
Ben Skywalker is taken to Marouk where he is beamed to a Kandy Kitchen, Mccue.
Ben Skywalker's body is placed in Karya.

# REFERENCES


[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language Models are Unsupervised Multitask Learners,” 2019.

[2] A. See, A. Pappu, R. Saxena, A. Yerukola, and C. D. Manning, “Do Massively Pretrained Language Models Make Better Storytellers?,” in *Conference on Computational Natural Language Learning (CoNLL)*, 2019, pp. 843–861.

[3] M. Alikhani, P. Sharma, S. Li, R. Soricut, and M. Stone, “Cross-modal Coherence Modeling for Caption Generation,” in *Annual Meeting ofthe Association for Computational Linguistics*, 2020, pp. 6525–6535, doi: 10.18653/v1/2020.acl-main.583.

[4] D. Ippolito, D. Grangier, C. Callison-Burch, and D. Eck, “Unsupervised Hierarchical Story Infilling,” in *Proceedings ofthe First Workshop on Narrative Understanding*, 2019, pp. 37–43.

[5] O. O. Marchenko, O. S. Radyvonenko, T. S. Ignatova, P. V. Titarchuk, and D. V. Zhelezniakov, “Improving Text Generation Through Introducing Coherence Metrics,” *Cybernetics and Systems Analysis*, vol. 56, no. 1, pp. 13–21, 2020, doi: 10.1007/s10559-020-00216-x.

[6] M. Abdolahi and M. Zahedi, “An overview on text coherence methods,” in *8th International Conference on Information and Knowledge Technology (IKT)*, 2016, no. 2, pp. 1–5, doi: 10.1109/IKT.2016.7777794.

[7] J. R. Hobbs, “Coherence and Coreference,” *Cognitive Science*, vol. 3, no. 1, pp. 67–90, 1979.

[8] B. J. Grosz and C. L. Sidner, “The structure of discourse structure,” *Computational Linguistics*, vol. 12, no. 3, pp. 175–204, 1985.

[9] B. J. Grosz, A. K. Joshi, and S. Weinstein, “Centering: A Framework for Modeling

the Local Coherence of Discourse," *Computational linguistics - Association for Computational Linguistics*, vol. 21, no. 2, pp. 203–225, 1995.

[10] P. W. Foltz, W. Kintsch, and T. K. Landauer, "The measurement of textual coherence with latent semantic analysis," *Discourse Processes*, vol. 25, no. 2–3, pp. 285–307, 1998, doi: 10.1080/01638539809545029.

[11] R. Barzilay and M. Lapata, "Modeling Local Coherence: An Entity-Based Approach," *Compuational Linguistics*, vol. 34, no. 1, pp. 1–34, 2008, doi: 10.1162/coli.2008.34.1.1.

[12] D. T. Nguyen and S. Joty, "A neural local coherence model," in *ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)*, 2017, pp. 1320–1330, doi: 10.18653/v1/P17-1121.

[13] L. Rimell, "Distributional lexical entailment by Topic Coherence," in *14th Conference of the European Chapter of the Association for Computational Linguistics 2014, EACL 2014*, 2014, pp. 511–519, doi: 10.3115/v1/e14-1054.

[14] N. Dziri, E. Kamalloo, K. W. Mathewson, and O. Zaiane, "Evaluating Coherence in Dialogue Systems using Entailment," in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019, p. 5.

[15] F. Rosner, A. Hinneburg, M. Röder, M. Nettling, and A. Both, "Evaluating topic coherence measures," in *Topic Models: Computation, Application and Evaluation Workshop (NeurIPS 2013)*, 2014, p. 4.

[16] C. Kiddon, L. Zettlemoyer, and Y. Choi, "Globally Coherent Text Generation with Neural Checklist Models," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016, pp. 329–339.

[17] T. Trabasso, T. Secco, and P. Van Den Broek, "Causal Cohesion and Story Coherence," in *Learning and Comprehension of Text*, H. Mandl, N. L. Stein, and T. Trabasso, Eds. 1984, pp. 83–111.

[18] A. C. Graesser, M. Singer, and T. Trabasso, “Constructing Inferences During Narrative Text Comprehension,” *Psychological Review*, vol. 101, no. 3, pp. 371–95, 1994.

[19] A. C. Graesser, K. L. Lang, and R. M. Roberts, “Question Answering in the Context of Stories,” *Journal of Experimental Psychology: General*, vol. 120, no. 3, pp. 254–277, 1991, doi: 10.1037/0096-3445.120.3.254.

[20] J. R. Meehan, “TALE-SPIN, An Interactive Program that Writes Stories,” in *5th International Joint Conference on Artificial Intelligence*, 1977, vol. 1, pp. 91–98.

[21] R. M. Young, S. G. Ware, A. Cassell, Bradly, and J. Robertson, “Plans and planning in narrative generation: a review of plan-based approaches to the generation of story, discourse and interactivity in narratives,” *Sprache und Datenverarbeitung, Special Issue on Formal and Computational Models of Narrative*, vol. 37, no. 1–2, pp. 41–64, 2013.

[22] M. O. Riedl and R. M. Young, “Narrative Planning: Balancing Plot and Character,” *Journal of Artificial Intelligence Research*, vol. 39, pp. 217–267, 2010.

[23] P. Xu *et al.*, “A cross-domain transferable neural coherence model,” in *57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 678–687, doi: 10.18653/v1/p19-1067.

[24] L. Logeswaran, H. Lee, and D. Radev, “Sentence Ordering and Coherence Modeling using Recurrent Neural Networks,” in *AAAI Conference on Artificial Intelligence*, 2018, pp. 5285–5292.

[25] J. Li and E. Hovy, “A model of coherence based on distributed sentence representation,” in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 2039–2048, doi: 10.3115/v1/d14-1218.

[26] R. Iida and T. Tokunaga, “Automatically Evaluating Text Coherence using Anaphora and Coreference Resolution,” in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2011, pp. 997–1006.

[27] T. Bohn, Y. Hu, J. Zhang, and C. X. Ling, “Learning Sentence Embeddings for Coherence Modelling and Beyond,” in *International Conference Recent Advances in Natural Language Processing (RANLP)*, 2019, pp. 151–160, doi: 10.26615/978-954-452-056-4_018.

[28] R. Porzel and I. Gurevych, “Contextual coherence in natural language processing,” *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, vol. 2680, pp. 272–285, 2003, doi: 10.1007/3-540-44958-2_22.

[29] P. Gervás, B. Díaz-Agudo, F. Peinado, and R. Hervás, “Story plot generation based on CBR,” *Knowledge-Based Systems*, vol. 18, no. 4–5, pp. 235–242, 2005.

[30] M. O. Riedl and V. Bulitko, “Interactive Narrative: An Intelligent Systems Approach,” *AI Magazine*, vol. 34, no. 1, pp. 67–77, 2013, doi: 10.1609/aimag.v34i1.2449.

[31] M. Cavazza, F. Charles, and S. J. Mead, “Planning characters’ behaviour in interactive storytelling,” *The Journal of Visualization and Computer Animation*, vol. 13, no. 2, pp. 121–131, 2002.

[32] J. Bates, “Virtual Reality, Art, and Entertainment,” *Presence: Teleoperators and Virtual Environments*, vol. 1, no. 1, pp. 133–138, 1992, doi: 10.1162/pres.1992.1.1.133.

[33] M. O. Riedl, A. Stern, D. M. Dini, and J. M. Alderman, “Dynamic Experience Management in Virtual Worlds for Entertainment, Education, and Training,” *ITSSA, Special Issue on Agent Based Systems for Human Learning*, vol. 4, no. 2, pp. 23–42, 2008.

[34] M. Riedl, C. J. Saretto, and R. M. Young, “Managing interaction between users and agents in a multi-agent storytelling environment,” in *Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS ’03)*, 2003, pp. 741–748, doi: http://doi.acm.org/10.1145/860575.860694.

[35] S. G. Ware and R. M. Young, “CPOCL: A Narrative Planner Supporting Conflict,” in *Seventh AAAI Conference on Artificial Intelligence and Interactive Digital*

*Entertainment (AIIDE)*, 2011, pp. 97–102.

[36] R. Farrell, S. G. Ware, and L. J. Baker, "Manipulating Narrative Salience in Interactive Stories Using Indexter's Pairwise Event Salience Hypothesis," *IEEE Transactions on Games*, vol. 12, no. 1, pp. 74–85, 2019, doi: 10.1109/tg.2019.2905230.

[37] J. Porteous, J. F. Ferreira, A. Lindsay, and M. Cavazza, "Extending Narrative Planning Domains with Linguistic Resources," in *AAMAS*, 2020, pp. 1081–1089.

[38] M. J. Nelson and M. Mateas, "Search-Based Drama Management in the Interactive Fiction Anchorhead," in *1st Annual Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005, pp. 99–104.

[39] P. W. Weyhrauch, "Guiding Interactive Drama," Carnegie Mellon University, Pittsburgh, PA, 1997.

[40] M. J. Nelson, D. L. Roberts, C. L. Isbell, M. Mateas, C. L. Isbell Jr., and M. Mateas, "Reinforcement learning for declarative optimization-based drama management," in *Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2006, pp. 775–782.

[41] B. Díaz-Agudo, P. Gervás, and F. Peinado, "A Case Based Reasoning Approach to Story Plot Generation," in *ECCBR 2004: Advances in Case-Based Reasoning*, 2004, vol. 3155, pp. 142–156.

[42] J. Zhu and S. Ontañón, "Shall I compare thee to another story?—An empirical study of analogy-based story generation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 2, pp. 216–227, 2014, doi: 10.1109/TCIAIG.2013.2275165.

[43] M. Lebowitz, "Planning Stories," in *Proceedings of the 9th Annual Conference of the Cogntive Science Society*, 1987, pp. 234–242.

[44] M. Mateas and A. Stern, "Integrating Plot, Character and Natural Language

Processing in the Interactive Drama Façade," in *Technologies for Interactive Digital Storytelling and Entertainment Conference*, 2003, vol. 2, pp. 139–151, doi: 10.1074/mcp.M700094-MCP200.

[45] R. E. Cardona-Rivera, B. A. Cassell, S. G. Ware, and R. M. Young, "Indexter: A Computational Model of the Event-Indexing Situation Model for Characterizing Narratives," *Working Notes from the Workshop on Computational Models of Narrative at the International Language Resources and Evaluation Conference*, pp. 32–41, 2012.

[46] B. Li, S. Lee-Urban, G. Johnston, and M. O. Riedl, "Story Generation with Crowdsourced Plot Graphs," in *27th AAAI Conference on Artificial Intelligence*, 2013, pp. 598–604.

[47] M. Roemmele, S. Kobayashi, N. Inoue, and A. M. Gordon, "An RNN-based Binary Classifier for the Story Cloze Test," in *2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, 2017, pp. 74–80.

[48] D. Thue, S. Schiffel, T. Þ. Guðmundsson, G. F. Kristjánsson, K. Eiríksson, and M. V. Björnsson, "Open World Story Generation for Increased Expressive Range," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10690 LNCS, pp. 313–316, 2017, doi: 10.1007/978-3-319-71027-3_33.

[49] R. Swanson and A. S. Gordon, "Say Anything: Using Textual Case-Based Reasoning to Enable Open-Domain Interactive Storytelling," *ACM Transactions on Interactive Intelligent Systems*, vol. 2, no. 3, pp. 1–35, 2012, doi: 10.1145/2362394.2362398.

[50] M. Guzdial, B. Harrison, B. Li, and M. O. Riedl, "Crowdsourcing Open Interactive Narrative," in *10th International Conference on the Foundations of Digital Games*, 2015, p. 9.

[51] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[52] R. Kiros *et al.*, “Skip-thought vectors,” in *Advances in neural information processing systems*, 2015, no. 786, pp. 3294–3302, doi: 10.1017/CBO9781107415324.004.

[53] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” in *International Conference on Learning Representations (ICLR)*, 2013, p. 12.

[54] U. Khandelwal, H. He, P. Qi, and D. Jurafsky, “Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context,” in *56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 284–294.

[55] A. Khalifa, G. A. B. Barros, and J. Togelius, “DeepTingle,” in *International Conference on Computational Creativity*, 2017, p. 8.

[56] M. Roemmele and A. S. Gordon, “Creative Help: A Story Writing Assistant,” in *International Conference on Interactive Digital Storytelling*, 2015, vol. 9445, pp. 81–92, doi: 10.1007/978-3-319-27036-4_8.

[57] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, “Convolutional Sequence to Sequence Learning,” *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 1243–1252, 2017, doi: 10.18653/v1/P16-1220.

[58] L. J. Martin *et al.*, “Event Representations for Automated Story Generation with Deep Neural Nets,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018, pp. 868–875.

[59] D. Ippolito, D. Grangier, D. Eck, and C. Callison-Burch, “Toward Better Storylines with Sentence-Level Language Models,” in *58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 7472–7478.

[60] A. Fan, M. Lewis, and Y. Dauphin, “Hierarchical Neural Story Generation,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018, pp. 889–898.

[61] L. Yao, N. Peng, R. Weischedel, K. Knight, D. Zhao, and R. Yan, “Plan-And-Write: Towards Better Automatic Storytelling,” in *AAAI Conference on Artificial Intelligence*, 2019, pp. 7378–7385.

[62] G. Chen, Y. Liu, H. Luan, M. Zhang, Q. Liu, and M. Sun, “Learning to Generate Explainable Plots for Neural Story Generation,” *IEEE/ACM Transactions on Audio Speech and Language Processing*, vol. 29, no. 4, pp. 585–593, 2021, doi: 10.1109/TASLP.2020.3039606.

[63] J. Shen, C. Fu, X. Deng, and F. Ino, “A Study on Training Story Generation Models Based on Event Representations,” in *International Conference on Artificial Intelligence and Big Data*, 2020, pp. 210–214.

[64] N. Mostafazadeh *et al.*, “A Corpus and Evaluation Framework for Deeper Understanding of Commonsense Stories,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 839–849.

[65] N. Mostafazadeh, M. Roth, A. Louis, N. Chambers, and J. F. Allen, “LSDSem 2017 Shared Task: The Story Cloze Test,” in *Linking Models of Lexical, Sentential and Discourse-level Semantics Workshop (LEDSem) at EACL*, 2017, pp. 46–51.

[66] M. Roemmele and A. Gordon, “Linguistic Features of Helpfulness in Automated Support for Creative Writing,” in *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018, pp. 14–19, doi: 10.18653/v1/w18-1502.

[67] E. Clark, A. S. Ross, C. Tan, Y. Ji, and N. A. Smith, “Creative Writing with a Machine in the Loop: Case Studies on Slogans and Stories,” in *23rd Intetnational Conference on Intelligent User Interfaces (IUI)*, 2018, pp. 329–340, doi: 10.1145/3172944.3172983.

[68] P. Ammanabrolu, W. Cheung, D. Tu, W. Broniec, and M. O. Riedl, “Bringing Stories Alive: Generating Interactive Fiction Worlds,” in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2020, vol. 16, pp. 3–9.

[69] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, “ALBERT: A Lite BERT for Self-Supervised Learning of Language representations,” in *Eighth International Conference on Learning Representations (ICLR)*, 2020, p. 17.

[70] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 3104–3112, doi: 10.1007/s10107-014-0839-0.

[71] D. Bamman, B. O’Connor, and N. A. Smith, “Learning Latent Personas of Film Characters,” in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, 2014, pp. 352–361.

[72] G. Prince, *A Dictionary of Narratology*. University of Nebraska Press, 1987.

[73] R. C. Schank and R. P. Abelson, *Scripts Plans Goals and Understanding*. Psychology Press, 1977.

[74] N. Chambers and D. Jurafsky, “Unsupervised Learning of Narrative Event Chains,” in *Association of Computational Linguistics*, 2008, vol. 94305, no. 14, pp. 789–797, doi: 10.1.1.143.1555.

[75] K. Pichotta and R. J. Mooney, “Learning Statistical Scripts with LSTM Recurrent Neural Networks,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, no. February, pp. 2800–2806.

[76] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. Mcclosky, “The Stanford CoreNLP Natural Language Processing Toolkit,” in *ACL System Demonstrations*, 2014, pp. 55–60.

[77] J. R. Finkel, T. Grenager, and C. Manning, “Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling,” in *Association for Computational Linguistics (ACL)*, 2005, pp. 363–370.

[78] G. A. Miller, “WordNet: A Lexical Database for English,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[79] K. Kipper Schuler, “VerbNet: A Broad-Coverage, Comprehensive Verb Lexicon,” University of Pennsylvania, 2005.

[80] K. Papineni, S. Roukos, T. Ward, and W. Zhu, “BLEU: a method for automatic evaluation of machine translation,” in *Association for Computational Linguistics (ACL)*, 2002, no. July, pp. 311–318, doi: 10.3115/1073083.1073135.

[81] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” in *arXiv:1603.04467*, 2016, p. 19.

[82] K. Pichotta and R. J. Mooney, “Using Sentence-Level LSTM Language Models for Script Inference,” in *Association for Computational Linguistics (ACL)*, 2016, pp. 279–289.

[83] R. A. Zwaan, M. C. Langston, and A. C. Graesser, “The Construction of Situation Models in Narrative Comprehension: An Event-Indexing Model,” *Psychological Science*, vol. 6, no. 5, pp. 292–297, 1995, doi: 10.1111/j.1467-9280.1995.tb00513.x.

[84] B. O’Neill and M. Riedl, “Dramatis: A Computational Model of Suspense,” in *AAAI Conference on Artificial Intelligence*, 2014, pp. 944–950.

[85] J. Niehaus and R. Michael Young, “Cognitive models of discourse comprehension for narrative generation,” *Literary and Linguistic Computing*, vol. 29, no. 4, pp. 561–582, 2014, doi: 10.1093/llc/fqu056.

[86] R. J. Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning,” *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[87] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second Edi. MIT Press, 1998.

[88] J. Li, W. Monroe, A. Ritter, and D. Jurafsky, “Deep Reinforcement Learning for Dialogue Generation,” in *Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1192–1202, doi: 10.1103/PhysRevB.94.020410.

[89] A. Y. Ng, D. Harada, and S. Russell, “Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping,” in *International Conference on Machine Learning (ICML)*, 1999, vol. 99, pp. 278–287.

[90] G. F. Jenks and F. C. Caspall, “Error on choroplethic maps: definition, measurement, reduction,” *Annals of the Assosiation of American Geographers*, vol. 61, no. 2, pp. 217–244, 1971.

[91] L. Litman, J. Robinson, and T. Abberbock, “TurkPrime.com: A versatile crowdsourcing data acquisition platform for the behavioral sciences,” *Behavior Research Methods*, vol. 49, no. 2, pp. 433–442, 2017.

[92] C. Purdy, X. Wang, L. He, and M. Riedl, “Predicting Generated Story Quality with Quantitative Metrics,” in *14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2018)*, 2018, pp. 95–101.

[93] P. Jain, P. Agrawal, A. Mishra, M. Sukhwani, A. Laha, and K. Sankaranarayanan, “Story Generation from Sequence of Independent Short Descriptions,” in *SIGKDD Workshop on Machine Learning for Creativity (ML4Creativity)*, 2017, p. 7, doi: 10.475/123.

[94] E. Clark, Y. Ji, and N. A. Smith, “Neural Text Generation in Stories Using Entity Representations as Context,” in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2018, pp. 2250–2260.

[95] N. Peng, M. Ghazvininejad, J. May, and K. Knight, “Towards Controllable Story Generation,” in *Proceedings of the First Workshop on Storytelling*, 2018, pp. 43–49.

[96] M. Roemmele and A. S. Gordon, “An Encoder-decoder Approach to Predicting Causal Relations in Stories,” in *Proceedings of the First Workshop on Storytelling*, 2018, pp. 50–59.

[97] P. Tambwekar, M. Dhuliawala, L. J. Martin, A. Mehta, B. Harrison, and M. O. Riedl, “Controllable Neural Story Plot Generation via Reinforcement Learning,” in

*International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 5982–5988.

[98] T. B. Hashimoto, K. Guu, Y. Oren, and P. Liang, "A Retrieve-and-Edit Framework for Predicting Structured Outputs," in *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018, p. 11.

[99] J. Gu, Z. Lu, H. Li, and V. O. K. Li, "Incorporating Copying Mechanism in Sequence-to-Sequence Learning," in *Association for Computational Linguistics (ACL)*, 2016, pp. 1631–1640.

[100] J. Pennington, R. Socher, and C. D. Manning, "GLoVe: Global Vectors for Word Representation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[101] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and Optimizing LSTM Language Models," in *International Conference on Learning Representations (ICLR)*, 2018, p. 13.

[102] T. Cazenave, "Monte Carlo Beam Search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 68–72, 2012.

[103] P. Anderson, B. Fernando, M. Johnson, and S. Gould, "Guided Open Vocabulary Image Captioning with Constrained Beam Search," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017, pp. 936–945.

[104] M.-L. Ryan, "Fiction, Non-Factuals, and the Principle of Minimal Departure," *Poetics*, vol. 9, no. 4, pp. 403–422, 1980.

[105] B. Magerko, J. E. Laird, M. Assanie, A. Kerfoot, and D. Stokes, "AI characters and directors for interactive computer games," *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, pp. 877–883, 2004.

[106] P. Clark, B. Dalvi, and N. Tandon, "What Happened? Leveraging VerbNet to

Predict the Effects of Actions in Procedural Text," in *arXiv preprint arXiv:1804.05435*, 2018, p. 9.

[107] A. K. Goel, S. Rugaber, and S. Vattam, "Structure, behavior, and function of complex systems: The structure, behavior, and function modeling language," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, vol. 23, no. 1, pp. 23–35, 2009, doi: 10.1017/S0890060409000080.

[108] H. Rashkin, M. Sap, E. Allaway, N. A. Smith, and Y. Choi, "Event2Mind: Commonsense Inference on Events, Intents, and Reactions," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018, pp. 463–473.

[109] N. Mostafazadeh *et al.*, "GLUCOSE: GeneraLized and COntextualized Story Explanations," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, vol. 4569–4586.

[110] H. Kwon *et al.*, "Modeling Preconditions in Text with a Crowd-sourced Dataset," 2020.

[111] J. Garbe, M. Kreminski, B. Samuel, N. Wardrip-Fruin, and M. Mateas, "StoryAssembler: An Engine for Generating Dynamic Choice-Driven Narratives," in *14th International Conference on the Foundations of Digital Games (FDG)*, 2019, p. 10, doi: 10.1145/3337722.3337732.

[112] P. Ammanabrolu *et al.*, "Story Realization: Expanding Plot Events into Sentences," in *AAAI Conference on Artificial Intelligence*, 2020, pp. 7375–7382.

[113] A. Holtzman, J. Buys, M. Forbes, and Y. Choi, "The Curious Case of Neural Text Degeneration," in *International Conference on Learning Representations (ICLR)*, 2020, p. 16.

[114] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019, pp. 4171–4186.

[115] P. Ammanabrolu *et al.*, "Guided Neural Language Generation for Automated Storytelling," in *Second Workshop of Storytelling (StoryNLP) at ACL*, 2019, p. 10.

[116] L. J. Martin, B. Harrison, and M. O. Riedl, "Improvisational Computational Storytelling in Open Worlds," in *9th International Conference on Interactive Digital Storytelling*, 2016, vol. LNCS 10045, pp. 73–84, doi: 10.1007/978-3-319-48279-8_7.

[117] L. J. Martin *et al.*, "Improvisational Storytelling Agents," in *Workshop on Machine Learning for Creativity and Design (NeurIPS 2017)*, 2017, p. 4.

[118] B. Levin, *English verb classes and alternations: A preliminary investigation*. University of Chicago Press, 1993.

[119] H. Zhang, Z. Liu, C. Xiong, and Z. Liu, "Grounded Conversation Generation as Guided Traverses in Commonsense Knowledge Graphs," in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020, pp. 2031–2043, doi: 10.18653/v1/2020.acl-main.184.

[120] S.-C. Lin, J.-H. Yang, R. Nogueira, M.-F. Tsai, C.-J. Wang, and J. Lin, "Conversational Question Reformulation via Sequence-to-Sequence Architectures and Pretrained Language Models," in *arXiv preprint arXiv:2004.01909*, 2020, p. 5.

[121] P. Wessa, "Free Statistics Software, Office for Research Development and Education, version 1.2.1," 2020. https://www.wessa.net/rwasp_cronbach.wasp/.

[122] L. J. Martin, S. Sood, and M. O. Riedl, "Dungeons and DQNs: Toward Reinforcement Learning Agents that Play Tabletop Roleplaying Games," in *Proceedings of the Joint Workshop on Intelligent Narrative Technologies and Workshop on Intelligent Cinematography and Editing*, 2018.

[123] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," in *NeurIPS Deep Learning Workshop*, 2013, p. 9.

[124] K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language Understanding for Text-

based Games Using Deep Reinforcement Learning," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015, pp. 1–11.

[125] M. Haroush, T. Zahavy, D. J. Mankowitz, and S. Mannor, "Learning How Not to Act in Text-Based Games," in *Workshop Track at ICLR 2018*, 2018, pp. 1–4.

[126] X. Yuan *et al.*, "Counting to Explore and Generalize in Text-based Games," in *Exploration in Reinforcement Learning Workshop at ICML 2018*, 2018, p. 12.

[127] L. Flekova and I. Gurevych, "Personality profiling of fictional characters using sense-level links between lexical resources," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015, pp. 1805–1816, doi: 10.18653/v1/d15-1208.

[128] O. J. Lee and J. J. Jung, "Modeling affective character network for story analytics," *Future Generation Computer Systems*, vol. 92, pp. 458–478, 2019, doi: 10.1016/j.future.2018.01.030.

[129] J. Porteous, F. Charles, and M. Cavazza, "Using Social Relationships to Control Narrative Generation," in *AAAI Conference on Artificial Intelligence*, 2015, pp. 4311–4312.

[130] E. Kim and R. Klinger, "An Analysis of Emotion Communication Channels in Fan Fiction: Towards Emotional Storytelling," in *Storytelling Workshop at ACL 2019*, 2019, p. 9.