Towards Wider Coverage Script Knowledge for NLP

Fangzhou Zhai

23rd November 2023

Dissertation

zur Erlangung des akademischen Grades eines Doktors der Philosophie
der Philosophischen Fakulät
der Universität des Saarlandes

Committee:

Prof. Dr. Ingo Reich

Prof. Dr. Alexander Koller

Prof. Dr. Dietrich Klakow

Prof. Dr. Vera Demberg

Dr. Mareike Hartmann

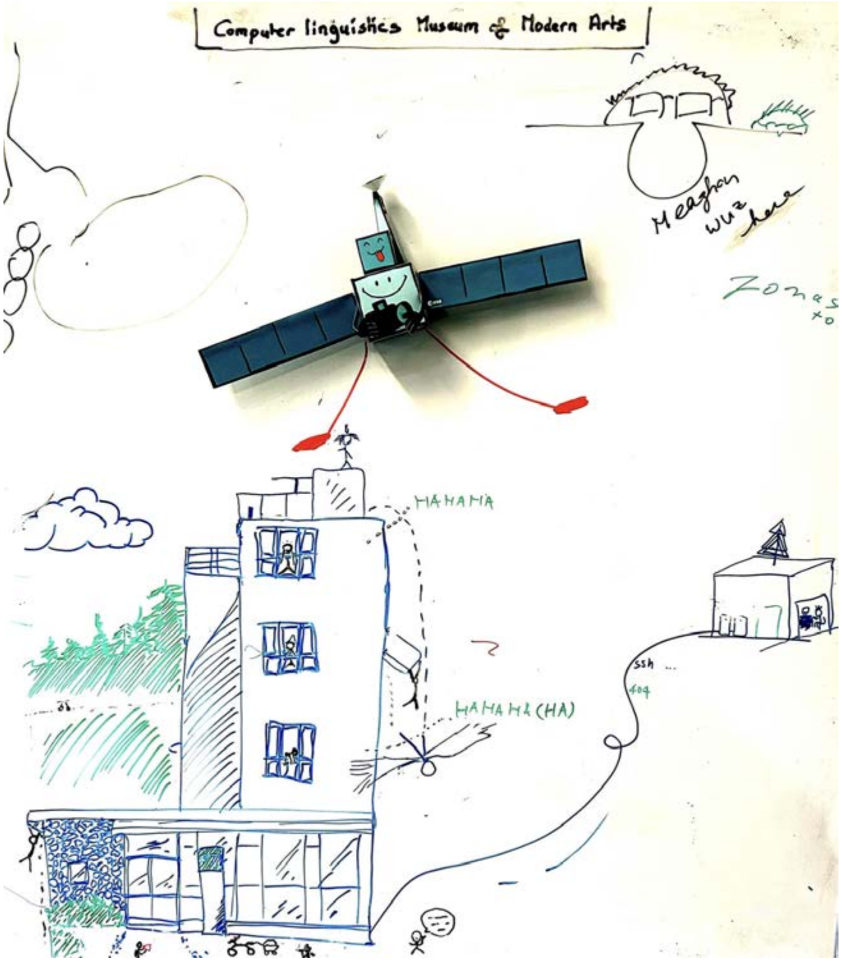
Data of Defense

23.11.2023

Dean

Dean: Prof. Dr. Stephanie Haberzettl


君子
不器


Computer linguistics Museum & Modern Arts
Meaghan wuz here
Zonas
xo
HAHAHA
HAHAHA (HA)
ssh ...
404

Abstract

This thesis focuses on acquiring wide coverage script knowledge.

Script knowledge constitutes a category of common sense knowledge that delineates the procedural aspects of daily activities, such as taking a train and going grocery shopping. It is believed to reside in human memory and is generally assumed by all conversational parties. Conversational utterances often omit details assumed to be known by listeners, who, in turn, comprehend these concise expressions based on their shared understanding, with common sense knowledge forming the basis. Common sense knowledge is indispensable for both the production and comprehension of conversation.

As outlined in Chapters 2 and 3, Natural Language Processing (NLP) applications experience significant enhancements with access to script knowledge. Notably, various NLP tasks demonstrate substantial performance improvements when script knowledge is accessible, suggesting that these applications are not fully cognizant of script knowledge. However, acquiring high-quality script knowledge is costly, resulting in limited resources that cover only a few scenarios. Consequently, the practical utility of existing resources is constrained due to insufficient coverage of script knowledge.

This thesis is dedicated to developing cost-effective methods for acquiring script knowledge to augment NLP applications and expand the coverage of explicit script knowledge. Previous resources have been generated through intricate manual annotation pipelines. In this work, we introduce automated methods to streamline the annotation process. Specifically, we propose a zero-shot script parser in Chapter 5. By leveraging representation learning, we extract script annotations from existing resources and employ this knowledge to automatically annotate texts from unknown scenarios. When applied to parallel descriptions of unknown scenarios, the acquired script knowledge proves adequate to support NLP applications, such as story generation (Chapter 6). In Chapter 7, we explore the potential of pretrained language models as a source of script knowledge.

Zusammenfassung

Skriptwissen

Als Kategorie des Allgemeinwissens umfasst Skriptwissen grundlegende Sachverhalte über unsere Welt. Genauer gesagt umfasst das Skriptwissen Fakten über alltägliche Handlungen wie das Zugfahren. Diese Fakten beinhalten die beteiligten Entitäten (die *Teilnehmer*), die Schritte zur erfolgreichen Durchführung der Handlung (die *Ereignisse*), die Muster, in denen sich diese Elemente verhalten, usw. Der Begriff *Skriptwissen* entstand aus frühen interdisziplinären Forschungen in den Bereichen Psychologie, künstliche Intelligenz und Linguistik. Ein *Skript* ist ein Sammlung von Wissen über eine Situation, mit der man häufig konfrontiert wird, wie zum Beispiel das Zugfahren; typischerweise finden in solchen Situationen bestimmte Handlungen sequenziell statt, 'wie in einem Skript'. Skriptwissen ist jedem allgemein bekannt und es wird angenommen, dass es im menschlichen Gedächtnis gespeichert ist (siehe Schank and Abelson, 1977).

Skriptwissen ist allgegenwärtig in der Verwendung natürlicher Sprache. Betrachten wir die Beispielgeschichte in Abbildung 1, in der der Erzähler seine Erfahrung des Auftankens eines Autos beschreibt. Beim Lesen der Geschichte können wir uns Gedanken über verschiedene relevante Details machen. Gab es einen Verkäufer an der Tankstelle? Wie hoch war der Preis nach dem Rabatt, wie viel hat es gekostet? Hat der Fahrer daran gedacht, dort nicht zu rauchen? Wir sind uns unsicher, was die Antworten auf diese Fragen sind, aber dennoch sind diese Fragen angemessen, denn beim Lesen von *mein Auto auftanken* wird sofort ein Wissenspaket im Kopf des Lesers aktiviert, das alle Elemente des Auftankens eines Autos enthält: das Auto, den Fahrer, die Tankstelle, das Erkennen eines niedrigen Tankstands, den Preis des Benzins, das Personal der Tankstelle, dass Rauchen an einer Tankstelle gefährlich ist, usw. Diese Informationen sind Allgemeinwissen, welches wir alle teilen. Genau diese Tatsache ermöglicht es uns, über alltägliche Handlungen zu sprechen, ohne den gesamten Prozess beschreiben zu müssen.

Skriptwissen ist unverzichtbar für das Verständnis natürlicher Sprache. Betrachten

Story: *Neulich fiel mir auf, dass ich nurnoch wenig Benzin im Tank hatte und dass ich mein Auto auftanken musste. Ich fuhr so in die Tankstelle ein, dass die Tanksäule auf meiner Fahrerseite war, da sich hier mein Tankdeckel befindet, und stellte das Auto ab. Die Tankstelle zu der ich gefahren war benutzt Bonuspunkte, die ich im Supermarkt bekommen hatte, also gab ich meine Kundennummer an und bekam 27 Cent pro Gallone erlassen, was super war! Ich zog meine Bankkarte durch um zu Bezahlen und öffnete den Tankdenkel mit einem Hebel in meiner Tür und schraubte den Tankdeckel ab. Ich* ***drückte den Griff*** *und ließ Benzin in den Tank bis es automatisch stoppte, da der Tank voll war. Ich entfernte vorsichtig den Zapfhahn vom Benzintank, plazierte ihn wieder auf der Zapfsäule, und schloß den Tankdeckel. Dann startete ich mein Auto um den Kilometerzähler zurückzusetzen und merkte mir den Stand, so dass ich sehen konnte wieviele Meilen mein Auto pro Gallone seit der letzten Füllung gemacht hatte.*

Frage: *Wann wurde der Zapfhahn in den Tank eingeführt?*
✓ *vor dem Befüllen mit Benzin*
✗ *nach dem Befüllen mit Benzin*

Figure 1: Ein Beispielfrage für die Bewertung von maschinellem Textverständnis aus Ostermann et al. (2019) (vom Autor ins Deutsche übersetzt). Die Frage bezieht sich auf das Auftanken eines Autos, und ihre Beantwortung erfordert Skriptwissen.

wir die Geschichte genauer. Zum Beispiel, *drückte* der Protagonist *den Griff* nachdem er *den Tankdeckel abgeschraubt hat*. Wir mögen ihm nicht viel Beachtung schenken, aber es gibt einen definiten Artikel im Satz, *der* Griff. Aber welcher Griff? Es wurde zuvor kein Griff erwähnt. Und warum wird der Griff *gedrückt*? Drücken ist keine typische Handlung, die normalerweise an einem Griff ausgeführt wird; es ist üblicher, einen Griff zu halten oder zu greifen. Natürlich hat ein menschlicher Leser keine Schwierigkeiten bei der Interpretation von *dem Griff*, denn er kennt sich mit dem Auftanken eines Autos aus; er weiß, dass der Benzinschlauch mit einem Griff verbunden ist, der eine Klappe hat, und dass man die Klappe öffnet, indem man den Griff *drückt*. Weil er *Skriptwissen* besitzt. Der Akt des Verstehens geschieht in der Realität so schnell, dass kaum zu bemerken ist, was hier alles passiert. Jemand, der jedoch nichts über das Auftanken eines Autos weiß, wird die Situation nicht verstehen können.

Als direkte Konsequenz müssen Anwendungen, die natürliche Sprache verarbeiten (NLP-Anwendungen), Kenntnis über Skriptwissen haben. Betrachten wir zum Beispiel die Frage in Abbildung 1: *Wann hat der Protagonist den Tankrüssel in den Tank gesteckt?* Es erscheint uns trivial. Natürlich sollte der Tankrüssel *vor* dem Befüllen des Tanks und nicht *danach* im Tank sein, da wir ein Auto betanken. Dennoch sind Fragen wie diese für modernste Textverständnismodelle schwer zu beantworten (das beste in dieser Arbeit untersuchte Modell erreicht eine Genauigkeit von 65 Prozent für Fragen, die mit 'wann' beginnen, siehe Ostermann et al., 2019). Ein Hauptgrund dafür ist, dass das Einführen des Zapfhahns in den Tank im Text nicht explizit erwähnt wird. Somit hat ein Modell ohne Skriptwissen keine Möglichkeit, die Verbindung zwischen dem, was im Text erwähnt wird, und diesem bestimmten verborgenen Ereignis herzustellen.

Skriptrepräsentation für NLP

Skriptwissen ist für verschiedene NLP-Anwendungen von Nutzen. Hier sind einige Beispiele: Die prototypische Beziehung zwischen Skriptereignissen und -teilnehmern kann Ambiguitäten bei der Korefernzauflösung eindeutig machen (Rahman and Ng, 2012). Das Vorhersehen zukünftiger Ereignisse hilft dabei, Diskursrelationen zu bestimmen (siehe z.B. Lee et al., 2020) (Abschnitt 2.4). Als eine Art prozedurales Wissen ermöglicht Skriptwissen darüberhinaus Anwendungen Textplanung durchzuführen, um Geschichtengenerierung zu steuern (siehe z.B. Zhai et al., 2019a) (Kapitel 3).

NLP-Anwendungen haben jedoch keinen ausreichenden Zugang zu Skriptwissen. Es gibt verschiedene Ansätze zur Repräsentation von Skriptwissen, die umfangreich untersucht wurden. Am intuitivsten ist es, das Skiptwissen direkt durch Sequenzen von Verben kodiert. Da Skriptwissen sich auf Sequenzen von Handlungen bezieht, kodieren typische Sequenzen von Verben oder Prädikat-Argument-Strukturen eine gewisse Menge an Skriptwissen. Zum Beispiel könnte ein typischer Restaurantbesuch durch eine Sequenz wie *ankommen(im Restaurant), setzen, bestellen(Essen), essen(Essen), bezahlen(Rechnung)* dargestellt werden.

Dieser Ansatz, der von Chambers and Jurafsky (2008,0) entwickelt wurde, erfasst die Intuition, dass spezifische Handlungen in Abhängigkeit der vorhergehenden erwartet werden können. Zum Beispiel findet das Bezahlen tendenziell nach dem Essen statt, daher könnte essen nach bestellen in einem Restaurant erwartet werden. Wenn jemand zum Flughafen geht und eincheckt, ist es wahrscheinlich, dass er das Flugzeug besteigt.

Andererseits könnten wir, im Gegensatz zur Darstellung als Abfolge spezifischer Prädikate, auch von der Textoberfläche abstrahieren und abstrakte Ereignis- (und Teilnehmer-) Typen entwickeln. Dies ist vorteilhaft, da äquivalente Handlungen für Skriptszenarien unterschiedliche Oberflächenrealisierungen aufweisen können. Man kann zu einem Restaurant *fahren*, in einem Restaurant *ankommen* oder ein Restaurant *besuchen*. Diese Verbphrasen sind im Allgemeinen nicht äquivalent, erfüllen jedoch die gleiche Rolle im Vorgang eines Restaurantbesuchs. Autoren wie Modi et al. (2016); Wanzare et al. (2017) sammeln parallele Beschreibungen von Skriptszenarien und annotieren die Texte mit Schlüsselelementen des Skriptwissens, wie Ereignissen und Teilnehmern, um Korpora wie *InScript* zu erstellen. Obwohl dies kostspielig ist, ermöglichen uns diese Annotationen ein einfaches Ablesen typischer Ereignissequenzen und die Erstellung von TSGs (Text Structured Graphs); das Skriptwissen wird explizit gemacht und ist somit für nachfolgende Anwendungen zugänglich.

Die Ansätze zur Repräsentation von Skriptwissen stehen vor verschiedenen technischen Herausforderungen, angesichts derer der letztere Ansatz besser geeignet ist, um Skriptwissen darzustellen.

Die erste Herausforderung besteht darin, dass Skriptwissen vom Oberflächentext oft versteckt ist. Dies liegt daran, dass Skriptwissen von allen Beteiligten in einer Konversation als Allgemeinwissen vorausgesetzt wird. Zum Beispiel sollte jeder, der das Säuglingsalter überwunden hat, wissen, dass er nach dem Essen in einem Restaurant bezahlt, ohne dass es ihm gesagt werden muss. Daher enthält eine Sequenz von Verben in natürlichsprachigen Texten, die effizient und informativ sind, bei Weitem nicht jedes Ereignis, das stattfindet, wenn ein Szenario instantiiert wird, da die meisten Ereignisse erwartet werden und weniger informativ sind. Man könnte sich über ein teures Restaurant beschweren, ohne zu erwähnen, dass man dort bestellt und gegessen hat. Tatsächlich sind die aus den meisten Texten extrahierten Prädikatsequenzen kleine Fragmente dessen, was tatsächlich in der physischen Welt stattgefunden hat. Vollständiges Skriptwissen über das Szenario von Interesse aus diesen fragmentierten Beschreibungen zu ziehen, ist eine schwierige Aufgabe. Im Gegensatz dazu sammeln Korpora wie *InScript* (Modi et al., 2016) mehrere explizite Beschreibungen jedes Skriptszenarios, für die den Annotatoren spezifische Anweisungen gegeben werden, um das Szenario im Detail zu erklären. Die Art und Weise, wie diese Texte gesammelt werden, ermöglicht es, dass viel weniger Inhalte aus dem Oberflächentext ausgelassen werden; außerdem werden Inhalte, die in der einen Beschreibung ausgelassen werden, oft in den anderen Beschreibungen explizit erwähnt, was es den Forschern ermöglicht, manche Ereignisse zu identifizieren, die zwar stattfinden, aber aus einigen der Beschreibungen ausgelassen werden.

Die zweite Herausforderung liegt in der Spezifität des Szenarios. Unterschiedliche Szenarien gehen mit unterschiedlichen Mustern in den Ereignissequenzen einher. Zum Beispiel wird bezahlen allgemein erwartet, wenn ein Essen voranging; jedoch spielt es eine Rolle, ob wir über das Szenario Restaurantbesuch oder Abendessen zu Hause sprechen. InScript ist szenariospezifisch, da die Texte szenarioweise gesammelt werden. Es ist jedoch schwierig, eine solche Spezifikation aus den aus natürlichen Texten extrahierten Prädikatsequenzen abzuleiten. Um welche Szenarien es sich handelt, kann man nur aus dem Text ableiten. Dies ist beispielsweise möglich, indem man das Thema des Textes bestimmt, aber selbst die Erkennung bereits bekannter Szenarien ist äußerst schwierig. Der aktuelle Stand der Technik erzielt hier weniger als 50 Punkte im F1-Maß auf Satzebene(siehe Wanzare et al., 2019).

Die Hauptproblematik des korpusbasierten Ansatzes zur Repräsentation von Skriptwissen liegt in der Abdeckung, d.h. dem Umfang des repräsentierten Skriptwissen. Die Erstellung von Korpora wie InScript ist teuer. Laut Modi et al. (2016) erfordert die Annotation neben dem Sammeln der Geschichten einen erheblichen manuellen Aufwand. Tatsächlich ist für jedes Szenario eine beträchtlicher Expertenaufwand erforderlich, um die Menge von Ereignissen und Teilnehmern festzulegen, welche für die vollständige Annotation benutzt werden. Infolgedessen schließt InScript nur eine Handvoll Szenarien ein. Dies ist das Problem der *Abdeckung*. Die begrenzte Abdeckung möglicher Szenarien bedeutet, dass solche Repräsentationen für praktische Anwendungen nicht ausreichend sind.

Das Ziel dieser Arbeit ist es, die Kosten für die Erfassung von Skriptwissen im InScript-Stil zu reduzieren, und dessen Abdeckung zu erweitern. Im Wesentlichen entwickeln wir automatische Methoden, um ähnliche Annotationen wie die in InScript, welche bisher manuell annotiert werden mussten, zu erzeugen.

Methode

Der Annotationsvorgang in InScript umfasst zwei Teilaufgaben. Diese bestehen darin, (1) die Menge der möglichen Ereignisse und Teilnehmer festzulegen und (2) Wörter zu identifizieren, die solche Annotationen hervorrufen. Um diesen Prozess zu automatisieren muss die Aufgabe des *Skript-Parsings* gelöst werden, bei dem gegeben einer Erzählung Wörter identifiziert werden, die Skript-Ereignisse und -Teilnehmer hervorrufen.

Skript-Parsing ist spezifisch für ein bestimmtes Szenario. Die Menge der Ereignisse und Teilnehmer ist für jedes Szenario unterschiedlich. Die Aufgabe des Skript-Parsings hat verschiedene Varianten, abhängig davon, wie viele Annahmen wir über das vorliegende Szenario machen. Im Rahmen des *überwachten Skript-Parsings* gehen wir von einer szenariospezifischen Überwachung aus, d. h. die Trainingsdaten und Testdaten beziehen sich auf dasselbe Szenario. Insbesondere sind uns die Mengen der möglichen Ereignisse und Teilnehmer bekannt. Die Aufgabe des überwachten Skript-Parsing wird als Sequenz-Labeling formuliert, d.h. dass jedes Wort in einer Sequenz einzeln annotiert wird (Ostermann et al., 2017; Zhai et al., 2021, see).

Um die Automatisierung von InScript-ähnlichen Annotationen zu ermöglichen wählen wir einen anderen Rahmen. Vor Allem möchten wir die Abdeckung von Skriptressourcen ohne umfangreiche manuelle Annotation erweitern, weshalb wir keine Kenntnis des Test-Szenarios voraussetzen können. Das bedeutet, dass wir keine Kenntnis über die Mengen der Ereignisse und Teilnehmer haben. Daher ergibt es keinen Sinn, das Skript-Parsing als Sequenz-Labeling zu formulieren. Stattdessen haben wir als Testeingabe eine Reihe von parallelen Beschreibungen des Test-Szenarios. Im Fall der Annotation von InScript handelt es sich dabei um etwa 100 Geschichten zu einem unbekannten Szenario. Das Skript-Parsing ohne szenariospezifische Überwachung ist eine *Clustering*-Aufgabe. Durch das Gruppieren von kontextualisierten Wörtern, die dasselbe Ereignis oder denselben Teilnehmer anzeigen, erfassen wir gleichzeitig die Mengen der Ereignisse und Teilnehmer, sowie die Zuordnung der Wörter zu den von ihnen hervorgerufenen Ereignissen oder Teilnehmern.

In dieser Arbeit formulieren wir das Skript-Parsing für unbekannte Szenarien als ein Zero-Shot-Lernproblem. Unsere Überlegung ist, dass ein Skript-Parser, obwohl keine szenariospezifische Überwachung gegeben ist, dennoch entscheidendes Wissen über das Skript-Parsing aus Daten lernen kann, die sich auf verschiedene Szenarien beziehen. Wir beobachten, dass verschiedene Aspekte des Skript-Parsings tatsächlich szenarioneutral sind: Es gilt für jedes Szenario, dass Entitäten innerhalb derselben Koreferenzkette denselben Teilnehmer hervorrufen sollten; für jedes Szenario folgen die Ereignisse einer zeitlichen Reihenfolge, die größtenteils sequenziell ist. Wir suchen nach einem Ansatz, der dieses Wissen von annotierten Szenarien auf ungesehene Szenarien überträgt.

Unsere Lösung ist ein Zero-Shot-Skript-Parser, der auf *unbekannten* Szenarien angewendet werden kann. Der Parser gibt, gegeben einer Menge von Erzählungen über ein Szenario, zu dem wir während der Trainingphase keine Informationen haben, die Mengen der Ereignisse und Teilnehmer aus, während er gleichzeitig die Wörter identifiziert, die Ereignisse oder Teilnehmer hervorrufen. Zu diesem Zweck entwickeln wir geeignete Repräsentationen, um den Clustering-Prozess zu ermöglichen. Die Repräsentation hat zwei Hauptbestandteile. Erstens betrachten wir allgemeine kontextualisierte Worteinbettungen, die von vortrainierten Modellen bereitgestellt werden. Unsere Hypothese ist, dass diese Einbettungen Merkmale

enthalten, die informativ für das Skript-Parsing sind; jedoch kodieren sie auch weniger relevante Merkmale. Zweitens transformieren wir den ursprünglich allgemeinen Einbettungsraum in einen spezialisierten Raum für das Skript-Parsing. Wir lernen diese Transformation basierend auf der im annotierten InScript verfügbaren Überwachung, welche aus Clustern von Oberflächentexten besteht, die dasselbe Ereignis oder denselben Teilnehmer hervorrufen. Wir transformieren die allgemeinen Einbettungen, um deren Konsistenz mit diesen Clustern zu verbessern, dahingehend, dass diese Cluster im transformierten Embedding-Raum eine höhere Reinheit aufweisen. Insbesondere sollten Instanzen, die im selben Cluster landen, eine höhere Ähnlichkeit aufweisen, während Instanzen aus verschiedenen Clustern eine geringere Ähnlichkeit aufweisen sollten.
Mit Einstellungen ähnlich derer, die bei der Annotation von InScript vorlagen, liegt die Leistung unseres Zero-Shot-Parsers bei durchschnittlich 70 Punkten im F1-Maß, was mit einem zuvor veröffentlichten überwachten Parser vergleichbar ist. Diese Leistung reicht aus, um die Erfassung von Skriptwissen zu ermöglichen. Wir extrahieren Skriptwissen aus MCScript, einem Korpus, dessen Sprache einfach und ähnlich zu unseren Trainingsdaten ist. Das von uns gewonnene Skriptwissen ist von annehmbarer Qualität, was die Erstellung von TSGs ermöglicht und NLP-Anwendungen wie der Geschichtenerzeugung gerecht werden könnte. Auf dem Spinn3r-Korpus, das aus natürlichsprachigen Texten besteht, liefert der Parser immer noch sinnvolle Cluster, und wir nehmen an, dass diese durch eine Vergrößerung der Menge an unannotierten Daten, die dem Parser zugeführt werden, weiter verbessert werden könnten.

Wir schließen den Hauptteil der Arbeit mit einer Verhaltensanalyse vortrainierter Sprachmodelle ab, um Licht auf ihr Potenzial als Quelle von Skriptwissen zu werfen. Unsere Untersuchungen stützen die Ansicht, dass modernste große Sprachmodelle als kompetente Quelle für Skriptwissen dienen könnten. Mit minimalem Input kann GPT-3 wesentlichen Elemente des Skriptwissens ausgeben und ein System zur Erzeugung von Skript-Geschichten ermöglichen, das eine bessere Leistung liefert als diejenigen, die wir basierend auf explizitem Skriptwissen entwickelt hatten. Diese Beobachtungen legen nahe, dass es aus Sicht von NLP-Anwendungen effizienter ist, sich, im Rahmen aller gebotener Vorsichtsmaßnahmen, auf ein großes Sprachmodell zu verlassen, welches scheinbar Kenntnis über gängiges Wissen besitzt, anstatt dieses explizit von Grund auf zu sammeln.

Acknowledgements

What a journey.
Eventually, I am on this page, given the opportunity to reflect on the past of time.

It was tough.

Trying to obtain a phD is not really that relaxing. For me, the worst part was mental health. Psychological issues had been accumulating in the depths of my mind. I had barely been able to maintain a fragile balance to cope with this burden and function as a unit of society. However, this illusion was suddenly shattered in the first month of my life as a PhD candidate. With a blink of an eye, there was no escape from these demons; I had the company of frustration, desperation, therapy, medication and their side effects. Now that the stage was set, the corona pandemic came, staging up the pressure on everyone's mental health with the chaos and isolation it incurs. What a mess.

It was heartening.

I am quite an introvert. I had kept everything to myself until it went beyond my capacity. The implication turned out to be a sight of countless helping hands.

Alexander Koller, my advisor, who was there throughout the entire journey. As suggested by the German word *Doktorvater*, I felt systematic efforts to support my growth as a researcher. When I asked him for my second month off to deal with my emotional issues, he was there. When I filled my papers with incomprehensible English, he was there. He would give me challenging research topics, watch me struggle, encourage me when I am about to flinch, and give his advice when it is most needed. He also secretly places a chocolate santa on everyone's table over Crhistmas. As strict as a father and as sweet as a father. With an advisor like him, I am happy to accept any challenge in my research.

Vera Demberg, who is also my advisor and a superwoman. For countless times, she lights up the path forward in the murkiest of my confusions; always energetic and cheerful, she handles everything swiftly and professionally, like a flagship that encourages me to climb up that path.

I shared an office with Iza Škrjanec in the darkest of my psychological disorders. I cannot thank her enough for the consolation she was during those couple of months.

I am in debt to everyone else in the group. Martin Villalba, Christoph Teichmann, Antoine Venant, Meaghan Fowlie, Jonas Groschwitz, Arne Köhn, Lucia Donatelli, Iris Ferrazzo, Mareike Hartmann, Sarubi Thillainathan and Yuekun Yao, they are friendly and resourceful. They are my debuggers, my German interpreters, my grammar checkers, my psychologists, my counselors, my dissert providers, my badminton coaches, etc.

I also received valuable research advice from other members of project A3. That would be Simon Ostermann, Lilian Wanzare, and Stefan Thater.

Then come my other cluster of friends in Saarland, Ni Dawei, Ni Ninwei, Shi Naosi, Wu Hao, Fan Yue, Guo Yanzhe, He Ziwei. Thanks for the company, food, and games. Being stupid together proved to be the best way to overcome pressures. ;)

My therapists, my family, and my old friends are all indispensable to prevent me from going completely insane.

Mister Klaus Erfort, his tranquil English garden and exquisite food always bring me peace of mind. Visiting his restaurant in early afternoon as the only customer is the best dining experience I have ever had, and more importantly, a convenient button that restores my peace of mind. Mister Hashimoto's restaurant is another such button that resets a troubled mind.

It was joyful.

As the drama of life unfolds and brings challenges and toughness, the people I met and the friendliness and support I read from their eyes shine in the darkest of nights, light up the path forward and warm my heart. It was in these lights that I learned to appreciate life per se. Thank you all.

Contents

Chapter 1

Introduction

Script Knowledge

This thesis focuses on the concept of *script knowledge*, which belongs to the category of commonsense knowledge that encompasses fundamental facts about our world. Specifically, script knowledge pertains to everyday activities, such as taking a train. It encompasses various details, including the entities involved (referred to as *participants*), the sequence of steps involved in carrying out the activity (known as *events*), and the behavioral patterns exhibited by these elements. The term *script knowledge* originated from early interdisciplinary research conducted in the fields of psychology, artificial intelligence, and linguistics. A *script* refers to a collection of knowledge about a commonly encountered situation, such as taking a train, where specific activities unfold sequentially, resembling a predefined script. Script knowledge is widely acknowledged and believed to be stored in human memory (see Schank and Abelson, 1977).

Story: *The other day I noticed that I was low on gas and had to fuel up my car . I pulled into the gas station with the pump on my driver 's side since that 's where my tank fills and turned off the car . The gas station I went to uses points that I earned at the grocery store so I entered my loyalty number and had 27 cents off a gallon which was great ! I swiped my debit card to pay and I popped the fuel door with a level inside my door and unscrewed the gas cap . I* ***squeezed the handle*** *and let the gas fill the tank until it automatically stopped because it was full . I carefully removed the nozzle from the fuel tank , replaced it on the pump and screwed the gas cap back on tightly and closed the fuel door . When I started my car to reset the trip meter and made note so I could see how many miles my car got per gallon since the last fill .*

Question: *When did they put the nozzle in their tank?*
✓ *before filling up with gas*
× *after filling up with gas*

Figure 1.1: An example machine comprehension question from Ostermann et al. (2019) about fueling up a car. The question requires script knowledge to answer.

Script knowledge is ubiquitous in the applications of natural language. Consider the example story in Figure 1.1, where the narrator describes his experience of fueling a car. As we read the story, various relevant details might appear in our mind. Was there a salesperson at the gas station? What was the price after the discount, and how much did it cost? Did the driver remember not to smoke there? We are not sure about the answers to these questions, yet these questions are appropriate, not surprising, because upon reading *fuel up my car*, a package of knowledge is immediately activated in the reader's mind, which contains all elements about fueling a car: the car, the driver, the gas station, noticing a low gas level, the price of the gas, the staff of the gas station, smoking is dangerous at a gas station, etc. This information is common knowledge among us. It is this very fact that allows us to talk about daily activities without describing the whole process.

Script knowledge is indispensable in the comprehension of natural language. Let us take a closer look at the story. For example, after *unscrewed the gas cap*, the protagonist *squeezed the handle*. It might not catch much of our attention, yet there is a definite article in the sentence, *the* handle. But which handle? No handle was mentioned before this phrase. And why is the handle *squeezed*? Squeezing is not a typical action that is generally performed on a handle; it is more common to hold or grab a handle. Of course, a human reader has no difficulty interpreting *the handle* because they know about fueling a car; they know that the gas tube is attached with a handle that has a switch on it and one opens the switch by *squeezing* the handle because they have *script knowledge*. The comprehension goes so quickly in reality that it is not easy to notice how much is going on here. However, anyone who knows nothing about fueling a car will not be able to understand the scenario. As a direct corollary, NLP applications need to be aware of script knowledge. Consider the question in Figure 1.1: *when did they put the nozzle in their tank*? At first glance, it may seem trivial. Of course, the nozzle should be in the tank *before* filling it rather than *after*, because we are fueling a car. However, questions like this one pose a challenge for state-of-the-art text comprehension models (the best model investigated in the paper performs at 65 percent accuracy on questions starting with 'when', see Ostermann et al., 2019). One important reason for this is that the event put the nozzle in the tank is not explicitly mentioned in the surface text. Consequently, a model without access to script knowledge lacks the means to establish the connection between the information mentioned in the surface text and this specific hidden event.

A most intuitive visualization of script knowledge is through the use of *Temporal script graphs (TSG)*. TSGs organize the *events* of an activity, known as the *scenario*, based on their temporal order of occurrence. TSGs provide a well-defined formalization and are easily accessible for NLP applications. Refer to Figure 1.2 for an illustrative example.

Script Representation for NLP

Script knowledge is valuable for various NLP applications, and we provide some examples here. The informative prototypical relationship among script events and participants aids in resolving ambiguities in coreference resolution (Rahman and Ng,

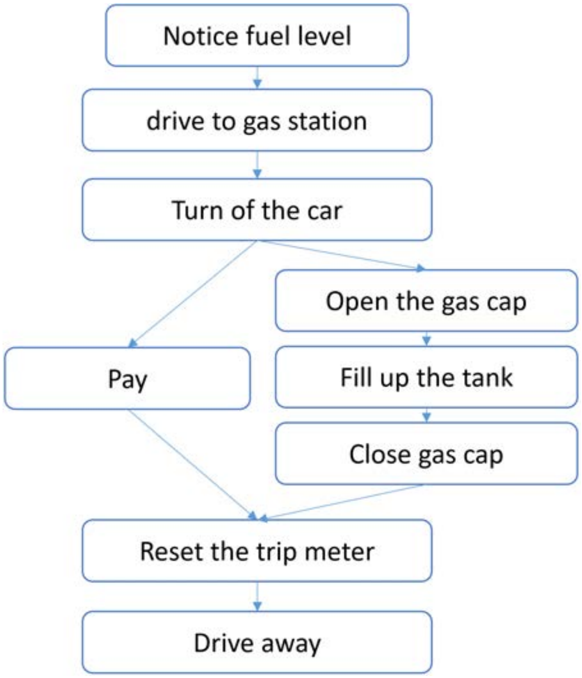


Figure 1.2: A temporal script graph of scenario fueling a car. The depiction connects events according to the temporal order in which they follow. Connections that could be inferred from transitivity are omitted from the depiction. The temporal order among the events is mostly sequential but not a total order.

2012). Anticipating upcoming events also assists in determining discourse relations (see, e.g., Lee et al., 2020) (Section 2.4). Furthermore, as a form of procedural knowledge, script knowledge enables applications to perform text planning for guiding story generation (see, e.g., Zhai et al., 2019a) (Chapter 3).

However, accessing script knowledge satisfactorily remains a challenge for NLP applications. Multiple plausible representations of script knowledge have undergone extensive investigation. The most intuitive representation involves sequences of verbs that straightforwardly encode script knowledge. Since script knowledge revolves around sequences of actions, typical sequences of verbs or predicate-argument structures encompass a certain amount of script knowledge. For example, a typical visit to a restaurant could be represented by a sequence such as *arrive(restaurant), sit, order(food), eat(food), pay(bill)*.

Pioneered by Chambers and Jurafsky (2008,0), this approach captures the intuition that specific activities can be anticipated based on preceding ones. For example, paying tends to occur after dining, and eating might be anticipated after ordering in a restaurant. If someone goes to the airport and checks in, it is likely that they will board the flight.

Secondly, as opposed to a sequence of specific predicates, we can also abstract away from the surface text and develop abstract event (and participant) types. This is desirable because equivalent actions for script scenarios can have different surface realizations. One can *drive to*, *arrive at*, or *visit* a restaurant. These verbal phrases are generally not equivalent but play the same role in the process of going to a restaurant. Authors like Modi et al. (2016); Wanzare et al. (2017) collect parallel descriptions of script scenarios and annotate the texts with key elements of script knowledge, such as events and participants, to establish corpora like *InScript*. Although more labor-intensive, these annotations allow us to easily identify typical event sequences and compile TSGs, making script knowledge explicit and accessible to downstream applications.

Representing script knowledge poses various technical challenges. Among these challenges, the latter approach is considered better.

One major challenge regarding script knowledge is that it is often hidden in the surface text. This is because script knowledge is commonly assumed to be shared by all participants in a conversation as common knowledge. For instance, it is generally expected that everyone knows they pay after eating in a restaurant without explicitly stating it. Consequently, naturally generated texts, which prioritize efficiency and informativeness, may exclude a sequence of verbs that encompasses every event occurring during the instantiation of a scenario, as most events are anticipated and considered less informative. It is not uncommon for someone to complain about an expensive restaurant without mentioning that they had ordered and eaten there. In reality, the predicate sequences extracted from most texts are merely fragments of the complete set of events that transpired. Acquiring comprehensive script knowledge about a particular scenario solely from these fragmentary descriptions proves to be a challenging task. In contrast, corpora such as *InScript* (Modi et al., 2016) compile multiple explicit descriptions for each script scenario, instructing workers to provide detailed explanations. The methodology employed in collecting these texts reduces the omission of contents from the surface text. Furthermore, contents omitted in one description are often explicitly mentioned in other descriptions, enabling researchers to recover some events that occurred but were excluded from certain narratives.

The second challenge is the specificity of scenarios. Different scenarios exhibit distinct patterns in event sequences. For instance, paying is generally anticipated following dining; however, it is crucial to differentiate between going to a restaurant and having dinner at home. InScript is scenario-specific, as it collects texts on a per-scenario basis. However, determining the scenario solely from sequences of predicates extracted from naturally generated texts proves to be difficult. One can only infer from the text which scenarios are being mentioned. This inference is possible, for example, by tracking the topic, but even detecting known scenarios has been shown to be challenging, with the state of the art achieving less than 50 points in sentence F1 (see Wanzare et al., 2019).

The primary concern regarding the corpus-based approach to script knowledge representation is its coverage. Acquiring corpora like InScript is an expensive endeavor. As Modi et al. (2016) state, apart from collecting the stories, the annotation process requires a significant amount of manual effort. Each scenario necessitates expert input to develop the set of events and participants for comprehensive annotation. Consequently, InScript was only able to include a limited number of scenarios. This limitation raises the issue of *coverage*. The constrained coverage of possible scenarios implies that it may not be suitable for practical applications.

The objective of this thesis is to reduce the cost of script knowledge acquisition in the style of InScript and expand its coverage. Essentially, we develop automatic methods to perform annotations similar to those in InScript, which were previously conducted manually.

Method

The annotation process of InScript involves two subtasks: (1) specifying the sets of events and participants, and (2) identifying tokens that evoke any of them. Automating this process gives rise to the task of *script parsing*, which, given a narrative, identifies tokens that evoke script events and participants. For instance, in the example depicted in Figure 1.3, a script parser labels tokens with the events (shown in blue) and participants (shown in orange) they evoke for a story about fixing a flat tire.

Script parsing is specific to the scenario of interest, as the sets of events and participants are unique to each scenario. The task of script parsing has different variants depending on the assumptions made about the scenario of interest. In the context of *supervised script parsing*, it is assumed that there is access to scenario-specific supervision, where the training and test data correspond to the same scenario. In this setting, the sets of events and participants are known. Supervised script parsing is formulated as a sequence labeling task (see Ostermann et al., 2017; Zhai et al., 2021).

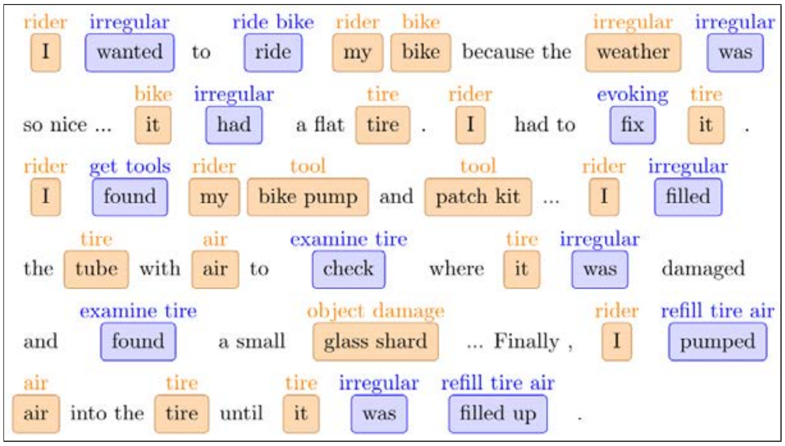
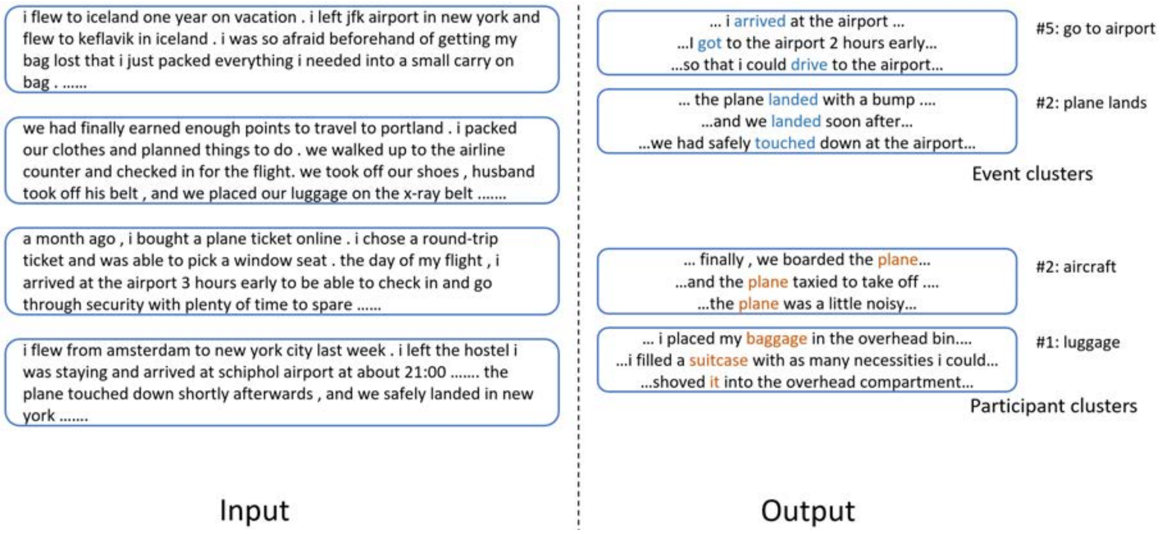


Figure 1.3: A story of fixing a flat tire from InScript. The annotation mainly performs two jobs. First, it specifies the set of participants and events; second, it assigns these labels to each token that evokes any of them. Automating this process makes the task of script parsing.

However, when automating InScript-style annotations, the settings are different. Importantly, the objective is to extend the coverage of script resources without relying heavily on manual annotation, and no prior knowledge of the test scenario is assumed. Therefore, access to the sets of events and participants is unavailable, making it inappropriate to formulate script parsing as a sequence labeling task. Instead, the test input consists of multiple parallel descriptions of the test scenario, such as approximately 100 stories on an unknown scenario. Script parsing without scenario-specific supervision becomes a *clustering* task. By grouping contextualized tokens that evoke the same event or participant, we simultaneously acquire the sets of events and participants, as well as the assignments between the tokens and the events or participants they evoke. Figure 1.4 illustrates script parsing as a clustering task, where the input includes multiple narratives on the scenario, with each token potentially evoking an event or a participant, and the output consists of sets of tokens that evoke the same event or participant.

In this thesis, we approach the task of script parsing on unknown scenarios as a zero-shot learning problem. Our rationale is that even without scenario-specific supervision, a script parser can still acquire crucial knowledge about script parsing from data on different scenarios. We observe that several aspects of script parsing are inherently scenario-neutral. For example, it holds true for any scenario that entities within the same coreference chain should evoke the same participant, and



i flew to iceland one year on vacation . i left jfk airport in new york and flew to keflavik in iceland . i was so afraid beforehand of getting my bag lost that i just packed everything i needed into a small carry on bag . ...

we had finally earned enough points to travel to portland . i packed our clothes and planned things to do . we walked up to the airline counter and checked in for the flight. we took off our shoes , husband took off his belt , and we placed our luggage on the x-ray belt ....

a month ago , i bought a plane ticket online . i chose a round-trip ticket and was able to pick a window seat . the day of my flight , i arrived at the airport 3 hours early to be able to check in and go through security with plenty of time to spare ......

i flew from amsterdam to new york city last week . i left the hostel i was staying and arrived at schiphol airport at about 21:00 .... the plane touched down shortly afterwards , and we safely landed in new york ....

... i arrived at the airport ...
...I got to the airport 2 hours early...
...so that i could drive to the airport...

... the plane landed with a bump ....
...and we landed soon after...
...we had safely touched down at the airport...

... finally , we boarded the plane...
...and the plane taxied to take off ....
...the plane was a little noisy...

... i placed my baggage in the overhead bin....
...i filled a suitcase with as many necessities i could...
...shoved it into the overhead compartment...

Figure 1.4: Script parsing without scenario-specific knowledge is a clustering task. Given parallel descriptions of the scenario of interest, tokens that evoke the same event or participant need to be identified and naturally form clusters of contextualized tokens.

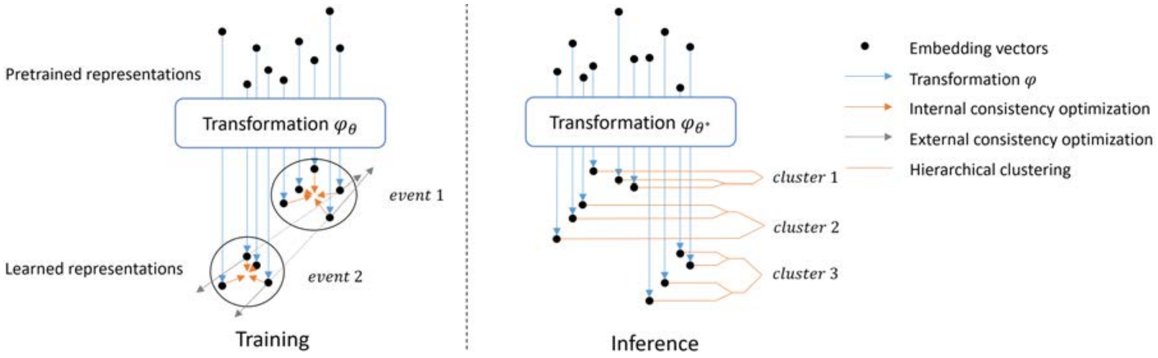


Figure 1.5: An illustration of our zero-shot script parser. In the training phase, we learn a representation that is consistent with the training set; during the inference phase, we perform clustering according to the representation to produce the clusters of tokens that evoke the same event or participant.

events within a scenario typically follow a mostly sequential temporal order. We aim to develop an approach that transfers this knowledge from annotated scenarios to unseen scenarios.

Our solution is a zero-shot script parser that operates on *unseen* scenarios. Given a set of narratives on a scenario for which we have no prior information during the training phase, the parser outputs the sets of events and participants, while identifying the tokens that evoke these events or participants. To achieve this, we develop appropriate representations to facilitate the clustering process. The representation consists of two main components. Firstly, we leverage general-purpose contextualized word embeddings provided by pre-trained models. Our hypothesis is that these embeddings contain informative features for script parsing, although they also encode less relevant features. Secondly, we transform the original general-purpose embedding space into a specialized space for script parsing. We learn this transformation using the available supervision from annotated InScript, which provides clusters of surface text that evoke the same event or participant. The transformation aims to improve the consistency of the general-purpose embeddings with these clusters, such that the clusters exhibit higher purity in the transformed embedding space. Specifically, instances assigned to the same cluster should exhibit higher similarities, while those from different clusters should not. Figure 1.5 illustrates the training and inference behaviors of our zero-shot parser.

Under settings similar to the annotation of InScript, our zero-shot parser achieves an average F1 score of around 70 points, which is comparable to a previous supervised parser. This performance is sufficient for acquiring script knowledge. We extract script knowledge from MCScript, a corpus with simple language that is similar to our training data. The acquired script knowledge exhibits decent quality, enabling the compilation of TSGs and accommodating NLP applications such as story generation. Even on the Spinn3r corpus, which comprises naturally generated texts, the parser produces meaningful clusters. We believe that the clustering performance could be further improved by increasing the amount of unannotated data fed into the parser. In conclusion, we perform a behavioral analysis of pre-trained language models to shed light on their potential as a source of script knowledge. Our explorations support the notion that state-of-the-art large language models can serve as competent sources of script knowledge. With minimal input, GPT-3 is capable of generating key elements of script knowledge and facilitating a script-story generation system that outperforms the ones we develop based on explicit script knowledge. These observations suggest, with appropriate cautions, that relying on a large language model that exhibits common-sense knowledge-aware behavior is more efficient for NLP applications than explicitly collecting such knowledge from scratch.

The Structure of the Thesis

Chapter 2 provides the contextual background for this thesis by discussing prior research on the representation of script knowledge within the NLP community. This chapter is divided into three sections: the efforts made by the NLP community in representing script knowledge, the availability of publicly accessible script-related resources, and empirical evidence showcasing how the incorporation of script knowledge enhances the performance of NLP applications.

Chapter 3 presents our work on script story generation, which involves generating narratives about script scenarios. We build upon an existing model that generates plausible narratives. However, the generated narratives tend to be simple in language and primarily focused on explicitly stating the events they are meant to depict. Our contribution lies in developing a model that incorporates richer details in the generated texts, resulting in more informative narratives. This effort aims to explore the extent to which text generation models can benefit from script knowledge. The results demonstrate that script knowledge enables the model to include more intricate details in a plausible manner. Nevertheless, the utility of this model is still constrained by the limited coverage of the script knowledge it has access to.

Chapter 4 and **Chapter 5** address the task of script parsing, which involves aligning natural language text with script knowledge by identifying the tokens that evoke script events and participants. Script parsing serves two critical purposes in script-related NLP: **(1)** it enables potential applications to access script knowledge that is mentioned in the surface text, and **(2)** it reveals additional patterns connecting script knowledge with surface text. Moreover, it can discover unknown event or participant classes when multiple descriptions of the same scenario are available.

Chapter 4 focuses on script parsing under supervised settings, where labeled data are available for all inference scenarios. Script parsing is traditionally formulated as a sequence-labeling task. Our approach involves a hierarchical modeling of the text. In addition to a sequence model operating at the token level, we incorporate another model to capture the sequence of events, which captures the procedural knowledge associated with the script. Our approach achieves a performance exceeding 80 points in F1 score, outperforming the previous state-of-the-art by almost 20 points.

Chapter 5 tackles the task of script parsing on unseen scenarios, where inference is performed on scenarios for which we have no access to labeled data. This parser enables the acquisition of script knowledge from unannotated texts. Although scenario-specific labeled data is not available, we believe that annotated corpora provide valuable insights into script parsing in general. Hence, we formulate the problem as a zero-shot parsing task, learning from annotated corpora and conducting inference on unannotated text from scenarios not included in the training data. Our approach primarily focuses on representation learning, embedding tokens that potentially evoke events or participants into a representation space optimized using the knowledge learned from annotated data.

Using the tools developed in **Chapter 5**, **Chapter 6** puts the acquired script knowledge without scenario-specific annotation into practice. We apply these tools to two unannotated corpora, namely MCScript and Spinn3r, mimicking the annotation process to acquire script knowledge. In addition to intrinsic evaluations, we revisit the task of story generation by training the model introduced in **Chapter 3** with automatically acquired script knowledge. While the quality of automatically acquired script knowledge is understandably lower than manually acquired knowledge, it proves to be sufficient for the story generation task examined in **Chapter 3**.

In **Chapter 7**, we engage in a discussion on the role of explicit script acquisition in NLP within the era of pre-trained language models. Our explorations suggest that, at the current stage of research, large language models appear to outperform explicit script knowledge methods, serving as both a source of script knowledge and a script-knowledge-aware agent.

List of Contributions and Collaboration

For clarity, we list the contents that we claim as part of the contribution of this thesis and clarify the participation of my collaborators. Of course, all of the following benefited from my advisors.

In Chapter 3, a story generation model that generates text with rich details. The method is presented in Section 3.3.

In Chapter 4, the supervised script parser. This chapter is based on Zhai et al. (2021) which is coauthored with, in addition to my advisors, Iza Škrjanec, who collected the data from the reverse translation (see Section 4.3.3) and performed most optimizations of the model variants that are subject to data augmentation. In Chapter 5, the zero-shot script parser. One of the probing tasks, sentence ordering, presented in section 5.5.2 was implemented by Katharina Stein.

Script acquisition on MCScript and Spinn3r described in Chapter 6. This chapter includes two small annotated datasets. Firstly, the annotation of the validation set and the test set of MCScript, which was performed by Katharina Stein after I developed the schema. Second, a small subset of Spinn3r is annotated for testing purposes.

In Chapter 7, the study of GPT-3's behaviors in sections 7.3 and 7.2.

Chapter 2

Background

2.1 Script Knowledge

Script knowledge is about how daily activities evolve. These activities are termed *scenarios*. A *scenario* is a situation of interest, such as fueling a car, taking a flight, obtaining a PhD, seeing a doctor, reviewing a paper, getting married, etc. A scenario could be any daily activity that occurs in a prototypical manner and is assumed to be common-sense knowledge. Regneri et al. (2011) were first to organize script knowledge around the concept of scenarios, as opposed to sequences of verbs or verbal phrases.

The granularity of script scenarios varies. One scenario could contain another, such as taking a train and buying a train ticket; they could partially overlap, such as getting a paper accepted and getting a paper rejected; and, of course, they could also be completely disjoint.

A specific process in which the activity takes place is termed an **instantiation** of the scenario. For example, a specific experience of fueling a car at my favorite gas station *instantiates* scenario fueling a car. The scenario is an abstraction of its instantiations; each scenario could be instantiated in different ways. To fuel a car, one could drop by a random gas station or go to his favorite one, pay with cash or credit card, etc.

An instantiation of any scenario consists of a sequence of activities. These activities are steps to accomplish the scenario; they are termed **events** of the scenario. For example, fueling a car may contain events such as noticing low fuel level, driving to the gas station, filling the tank, etc.

Events represent conceptual steps of the respective scenario, which abstract away from the surface texts. Therefore, an event could be **evoked** by various surface forms. For example, for event driving to the gas station, one might say *i pulled into the gas station*, or *I found a gas station nearby*, etc. These surface forms could be considered equivalent in terms of fueling a car. It also means that each event could see rather different surface realizations that are centered around different predicates. This is an important factor that makes it difficult to align script knowledge with surface texts.

Each specific instantiation of a scenario corresponds to an ordered chain of events, which we term an **agenda**.

It is also likely that events that actually took place do not get explicitly mentioned in the surface text. This is because a speaker or narrator tries to be informative with his utterances. Therefore, well-anticipated contents tend to be omitted as they are not informative. For example, when describing a restaurant experience, many would mention the food, the service or the atmosphere, however, we rarely hear people mentioning sitting down, as it is well anticipated. Actually, it is so well anticipated that it even takes some effort to notice that sitting down is missing from the surface text.

It is not trivial to determine exactly what counts as an event: if going to the train station counts as a step of taking a train, what about the activities that takes place over the course of going to the train station, like taking a taxi, taking a bus, or locating the train station? This concern is valid. Event granularity is a prominent technical challenge to script-related NLP. It will not escape our attention that many of these events can themselves be scenarios and consist of multiple steps, each of which could still be an independent scenario. going to train station itself can be complicated, which might include check train schedule, packing a luggage, etc.

A **participant** of a scenario is an entity that is likely to be involved in a typical instantiation of the scenario. In our example, the car, the driver, the gas station, and the nozzle are all participants. Similarly to events, participants could be evoked by different surface forms. These entities are termed participants because they participate in script events; these participants often take some important semantic role of the event-evoking verb. A participant might participate in multiple script events.

There is a special participant, the protagonist. The protagonist participates in most events, and is often the narrator of a personal experience, such as the driver in fueling a car, the passenger in taking a train, etc. The events that the protagonist participates could be sorted according to their temporal order to form a **narrative event chain** (see Schank and Abelson, 1977), a chain of events that share the same subject.

Some events and participants are not anticipated in every single instantiation of the respective scenario. In a narrative, we frequently encounter activities and entities that are not part of a prototypical instantiation of the scenario, but happen to be present in the specific instantiation. For example, one might mention *buying a drink* or *running into an old friend* in an taking a train experience. These events are not what we always anticipate from taking a train. We term these events **irregular events** and the participants **irregular participants**. Irregular events and participants are present in the surface text for specific communication reasons (after all, the narrator paid them enough attention to verbalize them), which diversifies the text itself. Their presence also reflects the fact that each scenario could be instantiated in different ways and under different circumstances.

2.2 Script Knowledge: Representation and acquisition

The *acquisition* of script knowledge develops methods that make script knowledge accessible to NLP applications. The *representation* of script knowledge is a specification of the data format that encodes script knowledge. Script knowledge has seen a variety of plausible representations proposed. Some of these are explicit and symbolic, where key elements of script knowledge could be directly read out, such as event chains, or annotated corpora like InScript where the sets of events and participants are simply spelled out; some others, like vector representations, are less about making it explicit to humans but more about establishing an agent that behaves in a script-knowledge-aware manner. In this section, we survey the efforts of the NLP community to acquire and represent script knowledge.

2.2.1 Temporal Script Graphs

As a type of procedural knowledge, script knowledge could be represented in the form of *temporal script graphs*, which link events with the *temporal order* in which they may take place. The events associated to each scenario follow some specific temporal order, which is a trivial fact, as some steps could only be undertaken after some others. Consider the scenario fueling a car. It is definite that the event driving to the gas station happens before filling the tank. The temporal order is tansitive and strictly asymmetric; it is not necessarily a total order, because some pairs of events could take place in arbitrary order. When fueling a car, we might pay before fueling the tank, or after. One might also reset the trip meter either before everything else or after, or whenever he has any time during the process. It is also possible that some drivers do not care about the trip meter at all. If we depict events as a directed graph according to their temporal order, we get a **temporal script graph** (Wanzare et al., 2017) of the scenario. See Figure 2.1 for an illustration.

Temporal script graphs could be conveniently acquired from corpora where tokens that evoke script events receive annotations, since the agendas could be read-off from these corpora, from which the temporal order could be estimated with count-based

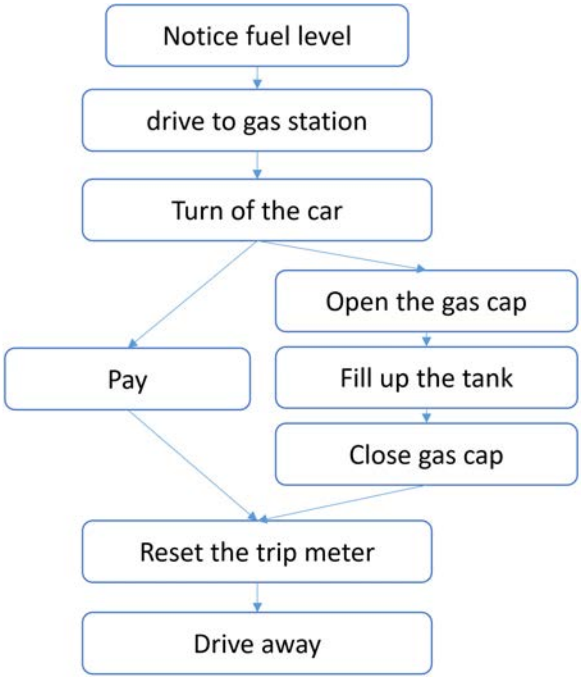


Figure 2.1: A temporal script graph of scenario fueling a car. Connections that could be inferred from transitivity are omitted from the depiction. The temporal order among events is mostly sequential, but not a total order.

methods.

2.2.2 Scripts as Planning Instances

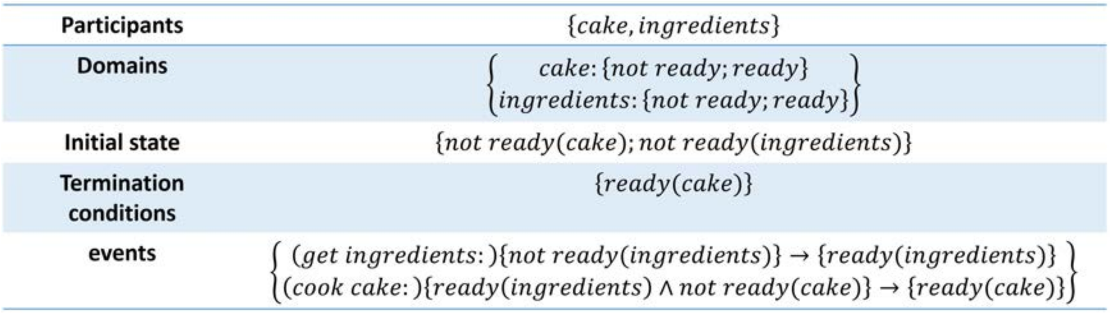
In their pioneering work, Schank and Abelson (1977) pointed out that script knowledge could be naturally formulated as instances of the *Planning* problem. *Planning* is a classical NP-complete problem that seeks a sequence of transitions that shift a system from its initial state to some goal state. One possible formalization is as follows.

Definition 2.2.1 (Planning)**.** *A planning problem is defined as a tuple* $\mathcal{P} = \{P, D, i, t, A\}$ *where*

> P *is a set of objects.*
>
> $D = \{d_p : p \in P\}$ *is the domain that specifies all possible world states. The world model consists of a set* P *of objects and a set* D *that collects the possible states* d_p *for each object* p*. Therefore, the Cartesian product* $\mathcal{S} = \prod_{p \in P} d_p$ *is the set of all possible world states.*
>
> $i \in \mathcal{S}$ *specifies the initial state.*
>
> $T \subset \mathcal{S}$ *specifies the goal states. It could be a singleton or a set of multiple goal states.*
>
> $A \subset \mathcal{S} \times \mathcal{S}$ *collects possible actions that could change the world state. For instances corresponding to realistic situations, it is very likely that, for some specific action, not all objects are relevant (for some intuition, you may pay for your gas regardless of whether the trip meter has been reset). Therefore, actions could often be succinctly packed into precondition / effect pairs.*

The answer would be a finite sequence of actions, aka a ***plan****, that leads the transition system from its initial state to some goal state.*

To explain how script knowledge could be formalized as an instance, let us consider an arbitrary scenario, for example, taking a train. Each instantiation corresponds to a sequence of events that take place in the physical world. Each possible event could only take place under certain circumstances (e.g. board with a ticket), whereas all it does is, in general, changing the state of the world. The sequence of events exhausts when the script is performed, that is, when the passenger arrives at the destination. We notice immediately that a sequence of actions also fits these descriptions. In addition, each scenario could be instantiated in different ways, resulting



Participants	$\{cake, ingredients\}$
Domains	$\left\{\begin{matrix} cake: \{not\ ready; ready\} \\ ingredients: \{not\ ready; ready\} \end{matrix}\right\}$
Initial state	$\{not\ ready(cake); not\ ready(ingredients)\}$
Termination conditions	$\{ready(cake)\}$
events	$\left\{\begin{matrix} (get\ ingredients:)\{not\ ready(ingredients)\} \rightarrow \{ready(ingredients)\} \\ (cook\ cake:)\{ready(ingredients) \wedge not\ ready(cake)\} \rightarrow \{ready(cake)\} \end{matrix}\right\}$

Figure 2.2: A brief version of baking a cake formulated as an instance of the planning problem. An event usually refers to a number of actions that have similar preconditions and effects.

in multiple sequences of events. These sequences are similar in the sense that they all accomplish the same thing: sending passengers to their destinations with the railway system. Every instantiation starts from the point when a passenger plans to travel; every instantiation ends when the passenger is at his destination.

Script knowledge naturally fits the formalization of planning problems. The world state could be described by specifying the state of all relevant entities, which includes all participants; each event is a set of actions that take place under similar circumstances to achieve similar things. We present an example in Figure 2.2.

The planning representation of script knowledge is very insightful. However, it might not be the most attractive approach for the purpose of informing NLP applications. The representation abstracts away from surface text and is symbolic and sensitive to noise (imagine one typo rendering the entire problem unsolvable). On the other hand, natural language is noisy and tends to omit descriptions of common-sense knowledge. That would mean aligning such formalization to surface text is highly nontrivial.

Many symbolic means of script representation could be seen as simplifications of the underlying planning instances. With the planning representation, we could precisely determine the temporal order between any pair of events, based on the preconditions and effects of all events, as it is in principle possible to enumerate all legitimate plans[1]. Another example is a sequence of predicate-argument structures, which

---

[1] Enumerating all plans in general is not guaranteed to be tractable. However, instantiations of script scenarios always see prototypical sequences of events, which suggest that script knowledge probably corresponds to a less complicated fragment of the planning problem.

exploits sequences such as $buy(passenger, ticket) \rightarrow board(passenger, train) \rightarrow arrive(train, station)$. Once again, these sequences could be inferred from the set of feasible plans.

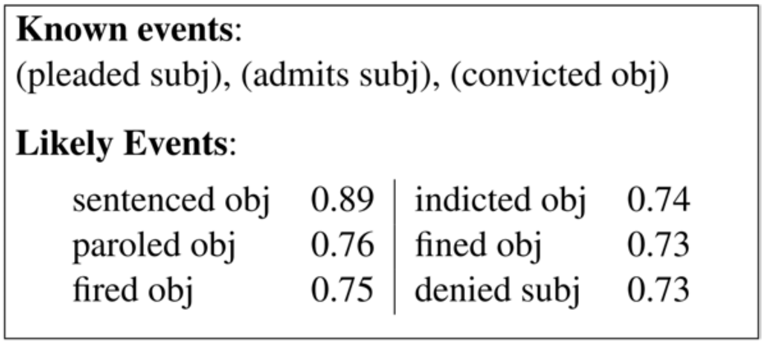
2.2.3 Narrative Event Chains

Automatic, large-scale script acquisition dates back to Chambers and Jurafsky (2008,0). Their work focuses on *narrative event chains*. A *narrative event* is defined as a tuple that includes an event (in most cases, represented by a verb) and its participants, such as *(board, passenger, bus)*, *(sentence, judge)*, etc. A **narrative event chain** is a partially ordered set of narrative events that share a participant. As an example, all activities of a specific person during a specific period of time form a narrative event chain.

Chambers and Jurafsky (2008) extracts sequences of events that share a common participant. These sequences are identified according to coreference information. With these sequences of events spread over multiple documents, the authors evaluate the connection between a pair of events with point-wise mutual information in terms of document-level cooccurrences. That is, an event e is considered to follow from e', if the existence of e on an average document significantly improves the probability of e' appearing in the same document. The authors propose a generative model of event sequences, in which, given a partially completed chain, the next event is selected as the one that has the maximum average PMI to those already in the chain. See Figures 2.3 and 2.4. The result is evaluated with an event cloze test, where the model needs to predict a missing event from a piece of text. Chambers and Jurafsky (2008) see substantial improvement over language model baselines.

The approach in Chambers and Jurafsky (2008) has seen multiple refinements. Jans et al. (2012) used bigram probabilities to construct narrative event chains, which yields better performance. Pichotta and Mooney (2014) went further to include multiple arguments for each event but uses a different notion of event chains in that they no longer constrain the events to share their protagonists. Rudinger et al. (2015) uses a predicate with its typed dependencies as the representation of events; in terms of sequence modeling, they used a log-bilinear language model that saw a noticeable performance improvement.

Narrative events align comfortably with surface text, and could be easily extracted. No annotations are necessary in the process, which means that such an approach scales well. Yet, there are important aspects of script knowledge that are not taken into account. First, these event sequences are not specific to the scenario



Known events:
(pleaded subj), (admits subj), (convicted obj)

Likely Events:

sentenced obj	0.89	indicted obj	0.74
paroled obj	0.76	fined obj	0.73
fired obj	0.75	denied subj	0.73

Figure 2.3: The initial segment of a narrative event chain and its possible successive events. Taken from Chambers and Jurafsky (2009).

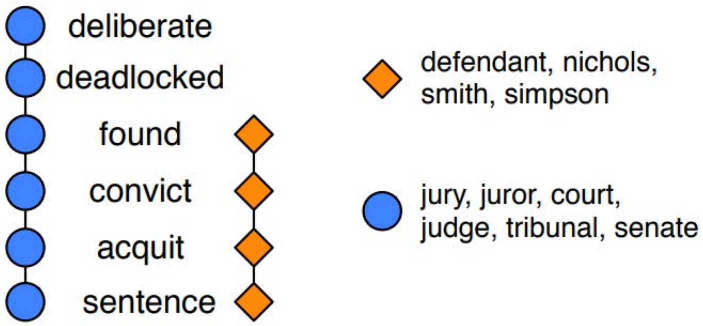


Figure 2.4: An illustration of two narrative schema that corresponds to two different shared participants. Taken from Chambers and Jurafsky (2009).

or the activity that each sequence of events are supposed to accomplish. Instead, they are mere sequences that are frequent in the corpus. Second, it is difficult to learn the events that are hidden from surface text, both because these events are not visible from surface text and because hidden events could be different given similar surface texts. For example, *gototrainstation* → *boardtrain* could be part of an event sequence corresponding to taking a train, or one corresponding to working as a conductor. Different patterns are expected from respective event sequences. Furthermore, different background scenarios would mean different hidden events: the passenger probably had made sure that he had a ticket, yet it is not the case for the conductor. In the next section, we will see that arranging script knowledge around the respective scenarios offers a neat solution.

2.2.4 Scenario-specific Script Acquisition

Limitations to event-chain-based approaches gradually becomes evident. As we have seen in Section 2.2.2 (see also Schank and Abelson (1977)), script knowledge about a certain activity could be formalized as an instance of planning. One implication is that these event chains are not mere event chains that appear frequently in some category of surface text, but sequences of actions that have a purpose. These chains of events must solve the planning problem and lead the system to a goal state; in other words, these events are there to perform a certain activity like baking a cake. Failure to take into account the goal of the event chain could cause confusion. For example, the first half of the event chains, respectively, corresponding to taking a flight and picking up a friend from airport, might be very similar. It is likely that the differences in the first half do not make their way into the surface text in a noticeable manner, in which case it would not be possible to correctly anticipate what events are coming next.

In order to take the activity of interest into account while modeling event sequences, Regneri et al. (2011) proposed the notion of *scenarios* (see also Section 1), which refers to the activity that a sequence of events is trying to accomplish. The introduction of scenarios initiated a line of research that investigates script knowledge in a scenario-aware manner. Modi et al. (2016); Wanzare et al. (2016) established corpora for scenario-specific script knowledge; Wanzare et al. (2017) investigates the temporal order of events within a scenario and compiled them into temporal script graphs (see Figure2.1). Ostermann et al. (2018a,0) are devoted to script-knowledge-based question answering.

A most important merit of the scenario-specific approach is allowing us to infer the events that are *hidden* from surface text. In naturally generated texts, one rarely, if ever, describes a scenario in full detail. A human, upon hearing about a script scenario like taking a train, is capable of retrieving the rest of the scenario from his knowledge base. Therefore, it would be difficult to extract the entire chain of events by investigating the verb sequences of natural texts, especially when the scenario is not specified. Modi et al. (2016); Wanzare et al. (2016) try to address this issue by collecting explicit representation of script knowledge. More on this in the next section.

2.2.5 Neural Script Representation

Script representation also saw the impact of neural network methodologies. Modi and Titov (2014); Granroth-Wilding and Clark (2016) learn vector representations of events by training them to correctly predict their temporal order using a linear ranker. Pichotta and Mooney (2016) use LSTM to model sequences of events in order to perform an event cloze test. Weber et al. (2018) use a variational autoencoder to generate sequences of events. In recognition of the hierarchical nature, the variational autoencoder is also designed to be hierarchical, which stochastically decides how the situation unfolds by generating a chain-shaped latent variable. Weber et al. (2020) model script knowledge and its relation to surface text according to Judia Pearl's causality framework(see, e.g., Pearl et al., 2000). The causal graph includes the event mentions, the text, and the discourse relations. This method saw a steady improvement over the point-wise mutual information criterion and the language model criterion.

Neural script representation methods considerably lift the requirement on manual inputs. As an example, randomly initialized vector representations in Modi and Titov (2014) could be trained to learn from the temporal order these events appear in the surface text and make predictions. The whole process does not require feature engineering. Moreover, neural methods could make use of general-purpose word embeddings, which, taken out-of-the-shelf, already contain much useful information. However, these methods do not grant wider coverage of available script resources. All these methods still require scenario-specific training supervision, which, obviously, grows in proportion with the number of scenarios we are interested in. This is exactly what hinders script knowledge representation from reaching a wider coverage.

2.2.6 Script Knowledge from Large Pre-trained Language Models

Large pre-trained language models, representatively BERT (Kenton and Toutanova, 2019), are trained with large amounts of texts. It is certain that commonsense knowledge leaves a trace in the distribution of their training data(we might see *bake a cake* but rarely *bake a train*). Therefore, we should be able to acquire script knowledge from these models. Sakaguchi et al. (2021) formalize script knowledge as directed acyclic graphs, whose nodes are all the events, and an edge is established from one event to another if there is a 'dependency' between them. Based on this formalization, they approach two tasks: edge prediction, where they use a two-stage classifier to determine whether there is an edge between two events and then the direction of the edge. The second task is scenario generation, for which they train a sequence-to-sequence model that takes the scenario name and number of events as input (for example, taking a train, in 3 steps) and outputs the list of events and edges. Both tasks are approached from fine-tuning large pre-trained language models. The performance on the edge prediction task is above 70 points F1 score, and generates from ROCStories(Mostafazadeh et al., 2016) to DeScript; the scenario generation task is evaluated with the edit distance between the output graph and its reference, which is around 5 while the average number of events in the reference graphs is 5.5.

Madaan et al. (2022) extract script knowledge from large language models trained for code, under the hypothesis that these models should be better at processing structured common sense knowledge as its input is structured. They also approach the two tasks mentioned above and find that the code LM outperforms natural language LM by a large margin.

Apart from establishing explicit representations for key elements of scripts such as events and participants, neural script acquisition could go more implicit. That is, instead of specifying how script knowledge looks like, we could also develop our models in such a way that it behaves in a script-knowledge-aware manner. In Chapter 7, we discuss how much state-of-the-art pretrained language models could offer from this perspective.

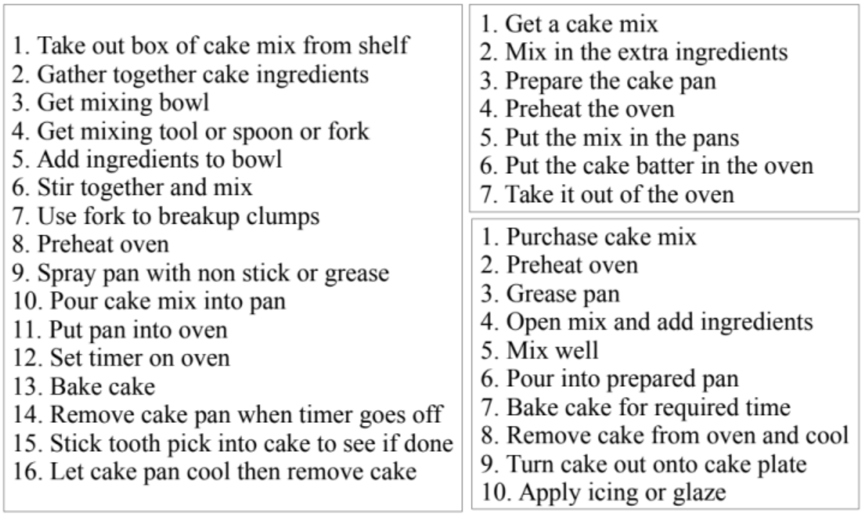
2.3 Resources

In this section, we go through some publicly available resources for script knowledge.

2.3.1 DeScript

DeScript is constructed by Wanzare et al. (2016). Instead of narratives, it crowdsources *event sequence descriptions.* Each event sequence description consists of a sequence of *event descriptions*, which is a short verbal phrase that specifies the event. For example, *buy a train ticket.* Therefore, each event sequence description corresponds to a specific instantiation of the scenario.

DeScript collects around 100 event sequence descriptions for each of 40 scenarios. These event sequence descriptions vary in their lengths, granularity, choice of verbs, etc. See Figure 2.5 for some examples of ESD. For each of those 40 scenarios,

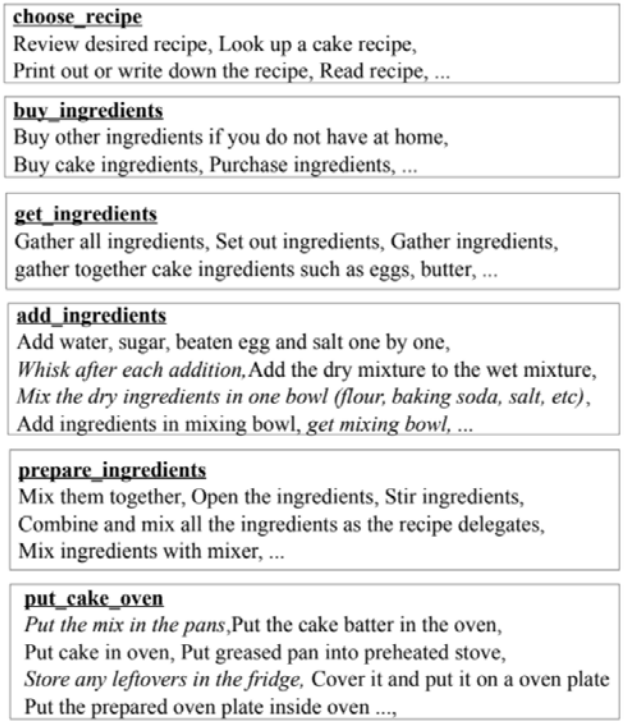


1. Take out box of cake mix from shelf
2. Gather together cake ingredients
3. Get mixing bowl
4. Get mixing tool or spoon or fork
5. Add ingredients to bowl
6. Stir together and mix
7. Use fork to breakup clumps
8. Preheat oven
9. Spray pan with non stick or grease
10. Pour cake mix into pan
11. Put pan into oven
12. Set timer on oven
13. Bake cake
14. Remove cake pan when timer goes off
15. Stick tooth pick into cake to see if done
16. Let cake pan cool then remove cake

1. Get a cake mix
2. Mix in the extra ingredients
3. Prepare the cake pan
4. Preheat the oven
5. Put the mix in the pans
6. Put the cake batter in the oven
7. Take it out of the oven

1. Purchase cake mix
2. Preheat oven
3. Grease pan
4. Open mix and add ingredients
5. Mix well
6. Pour into prepared pan
7. Bake cake for required time
8. Remove cake from oven and cool
9. Turn cake out onto cake plate
10. Apply icing or glaze

Figure 2.5: Examples of event sequence descriptions taken from Wanzare et al. (2016)



choose_recipe
Review desired recipe, Look up a cake recipe,
Print out or write down the recipe, Read recipe, ...

buy_ingredients
Buy other ingredients if you do not have at home,
Buy cake ingredients, Purchase ingredients, ...

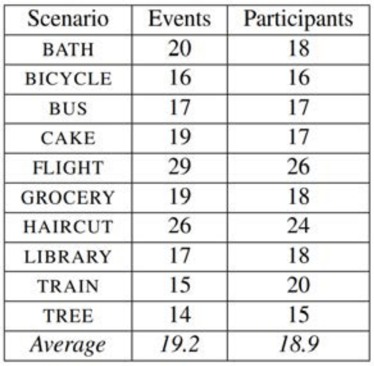
get_ingredients
Gather all ingredients, Set out ingredients, Gather ingredients,
gather together cake ingredients such as eggs, butter, ...

add_ingredients
Add water, sugar, beaten egg and salt one by one,
Whisk after each addition, Add the dry mixture to the wet mixture,
Mix the dry ingredients in one bowl (flour, baking soda, salt, etc),
Add ingredients in mixing bowl, *get mixing bowl,* ...

prepare_ingredients
Mix them together, Open the ingredients, Stir ingredients,
Combine and mix all the ingredients as the recipe delegates,
Mix ingredients with mixer, ...

put_cake_oven
*Put the mix in the pans,*Put the cake batter in the oven,
Put cake in oven, Put greased pan into preheated stove,
Store any leftovers in the fridge, Cover it and put it on a oven plate
Put the prepared oven plate inside oven ...,

Figure 2.6: Examples clusters of event descriptions about baking a cake, taken from Wanzare et al. (2016). Each set contains verbal phrases that are considered similar with regard to the scenario. In italics are some hard cases.



Scenario	Events	Participants
BATH	20	18
BICYCLE	16	16
BUS	17	17
CAKE	19	17
FLIGHT	29	26
GROCERY	19	18
HAIRCUT	26	24
LIBRARY	17	18
TRAIN	15	20
TREE	14	15
Average	*19.2*	*18.9*

Figure 2.7: The number of events and participant types of each InScript scenario, taken from Modi et al. (2016). A larger set of events or participants suggest more complexity within the scenario.

the corresponding event sequence descriptions are aligned with each other to link event descriptions that are believed to evoke the same event. This is essentially a clustering process. Wanzare et al. (2016) performed the alignment in a semi-supervised manner. The seeds consist of two parts: a small number of gold-standard seeds annotated by experts and extra seeds annotated through crowd-sourcing. The alignment is partial, i.e. not all event descriptions are required to be aligned with another. The outcome, after moderate post-processing, is sets of verbal phrases that are supposed to evoke the same events. To some extent, these are paraphrase sets. The set of events associated with the scenarios of interest, as well as the potential temporal order among them, is also acquired in the meanwhile. See Figure 2.6.

2.3.2 InScript

DeScript focuses on short event descriptions. *InScript* (Modi et al., 2016), in contrast, collects explicit script narratives and annotates them with elements of script knowledge, including script events, participants, etc. The establishment of InScript is a complex process. It comprises the following steps.

1.Scenario selection. InScript collect stories on 10 script scenarios. These scenarios make a subset of DeScript scenarios and are meant to be of different complexities and specificities. Figure 2.7 gives the set of all scenarios. In general, more complicated scenarios and less specific scenarios result in larger sets of events and participants.

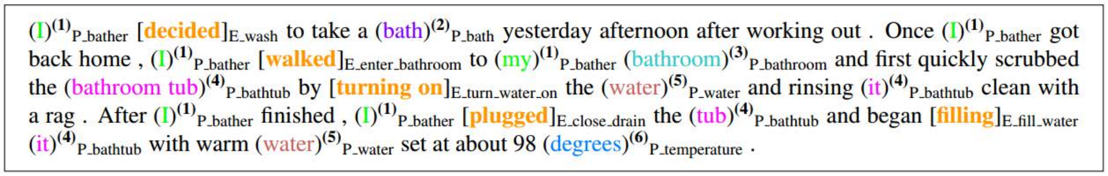
2.Story collection. InScript aims to collect explicit descriptions of each scenario of interest. However, script knowledge is a category of commonsense knowledge which is usually assumed by an author and remains hidden in surface text. The stories are crowd-sourced accordingly. To make script knowledge more explicit, the authors, after several pilot experiments, instruct the workers to *describe a scenario in the form of a story* ***as if explaining it to a child***. The result is relatively explicit descriptions of the respective scenarios in simple language[2]; besides, the activities in the stories roughly follow their temporal order. *InScript* contains around 100 crowd-sourced stories for each of the 10 scenarios. On average, there are 12 sentences per story, each consisting of 217 words. See Figure 2.9 for more statistics.

3.Developing Annotation Schema. A most crucial step in the pipeline is determining the sets of events and participants, which the authors term the *template* (see Figure 2.10). The preliminary templates are designed manually by the authors; later, the authors annotate 5 random stories per scenario to adjust the templates. The annotation per se also consists of multiple phases, and after the first phase some of these templates are adjusted again. We remark here that this step incurs a considerable amount of workload directly on the authors, which scales up with the number of scenarios. This is one of the main reasons that an InScript style corpus cannot include a large number of scenarios.

4.Annotation. With the templates developed, the main body of the annotation could start. Annotations are performed by undergraduate students of computational linguistics. During the annotation, each verb receives an event label and each NP head receives a participant label. Finally, all noun phrases receive a coreference chain annotation, indicating to which referent they refer. It is not hard to notice that not all NPs evoke a participant that is relevant to the scenario. For example, one fixing a flat tire story begins with *The weather was nice.* Apparently, *weather* is not relevant to fixing a flat tire. Likewise, some verbs simply refer to the incidents taking place within the course of the story, but are not indispensable for the scenario. Therefore, the tag sets also contain labels for irregular events and participants. Irregular event labels include '*unrelated* (not related to the main activity), *non-script* (related but not part of the activity per se), *unknown*' and *other*; irregular participant labels include *'other'*, *'suppvcomp'* (support verb complement), *'unclear'*,and *'head*

---

[2] It turns out that a considerable proportion of script events that actually take place are still omitted from the stories. For example, in taking a flight, few stories mention retrieving the luggage even if they mention checking in the luggage. But these stories are sufficiently explicit for script acquisition.



(I)$^{(1)}_{\text{P\_bather}}$ [decided]$_{\text{E\_wash}}$ to take a (bath)$^{(2)}_{\text{P\_bath}}$ yesterday afternoon after working out . Once (I)$^{(1)}_{\text{P\_bather}}$ got back home , (I)$^{(1)}_{\text{P\_bather}}$ [walked]$_{\text{E\_enter\_bathroom}}$ to (my)$^{(1)}_{\text{P\_bather}}$ (bathroom)$^{(3)}_{\text{P\_bathroom}}$ and first quickly scrubbed the (bathroom tub)$^{(4)}_{\text{P\_bathtub}}$ by [turning on]$_{\text{E\_turn\_water\_on}}$ the (water)$^{(5)}_{\text{P\_water}}$ and rinsing (it)$^{(4)}_{\text{P\_bathtub}}$ clean with a rag . After (I)$^{(1)}_{\text{P\_bather}}$ finished , (I)$^{(1)}_{\text{P\_bather}}$ [plugged]$_{\text{E\_close\_drain}}$ the (tub)$^{(4)}_{\text{P\_bathtub}}$ and began [filling]$_{\text{E\_fill\_water}}$ (it)$^{(4)}_{\text{P\_bathtub}}$ with warm (water)$^{(5)}_{\text{P\_water}}$ set at about 98 (degrees)$^{(6)}_{\text{P\_temperature}}$.

Figure 2.8: A fragment of InScript as is presented in the original paper. Tokens that evoke script events or participants receive respective annotations. Referring expressions along the same coreference chain receive the same color.

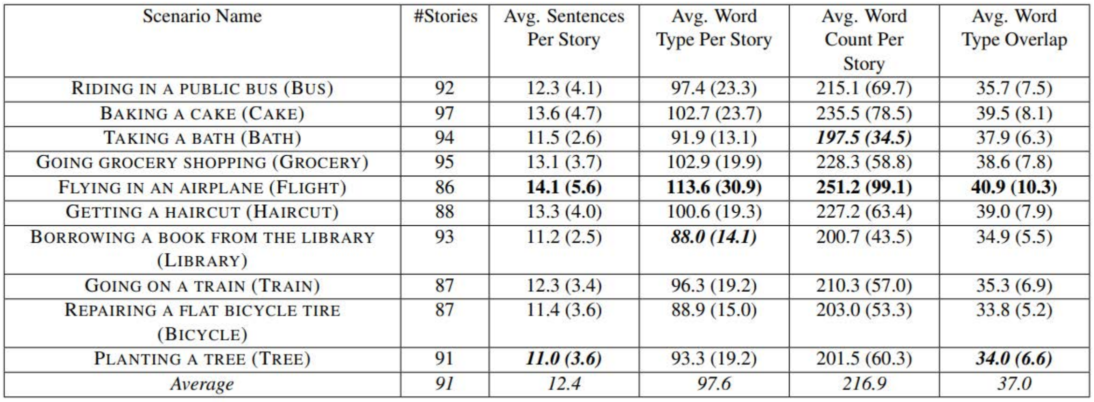
of partitive'. About 40% of the events and 20% of the participants are irregular. In total, the corpus contains 234k tokens, of which 15k received an event label, while 47k received a participant label. See Figure 2.8 for an illustration of the annotations. On average, there are 7 coreference chains per story with an average length of 6 tokens.

Inter-annotator agreements. 30 stories from 6 scenarios are randomly selected and annotated by 4 annotators to evaluate inter-annotator agreement, the results are given in Figure 2.11. Coreference annotations admit high inter-annotator agreement, with 90.5% coreference pairs agreed by at least 3 out of 4 annotators. The inter-annotator agreement for events and participants is, in comparison, moderate (the Fleiss' Kappa is at a moderate 0.64 for event labels and 0.77 for participant labels). In fact, the identification of event and participant evoking tokens is a more complicated task than identifying coreference pairs. The moderate inter-annotator agreements indicate that the task packs considerable difficulty for humans, even though the annotators are students of computational linguistics. The main effort of this thesis is to automate this complex InScript annotation process, without scenario-specific training supervision, to allow script knowledge to be presented in the form of figure 2.8 with a wide coverage.

2.3.3 MCScript

MCScript is a machine comprehension dataset proposed by Ostermann et al. (2018a) (Ostermann et al., 2018b, see also). A refined version, named *MCScript 2.0*, is later published by Ostermann et al. (2019). The purpose is to evaluate the potential contribution of script knowledge to text understanding.

MCScript2.0 is based on about 20 InScript-style stories for each of 200 scenarios; these scenarios are selected to reach a reasonable coverage of daily activities. These



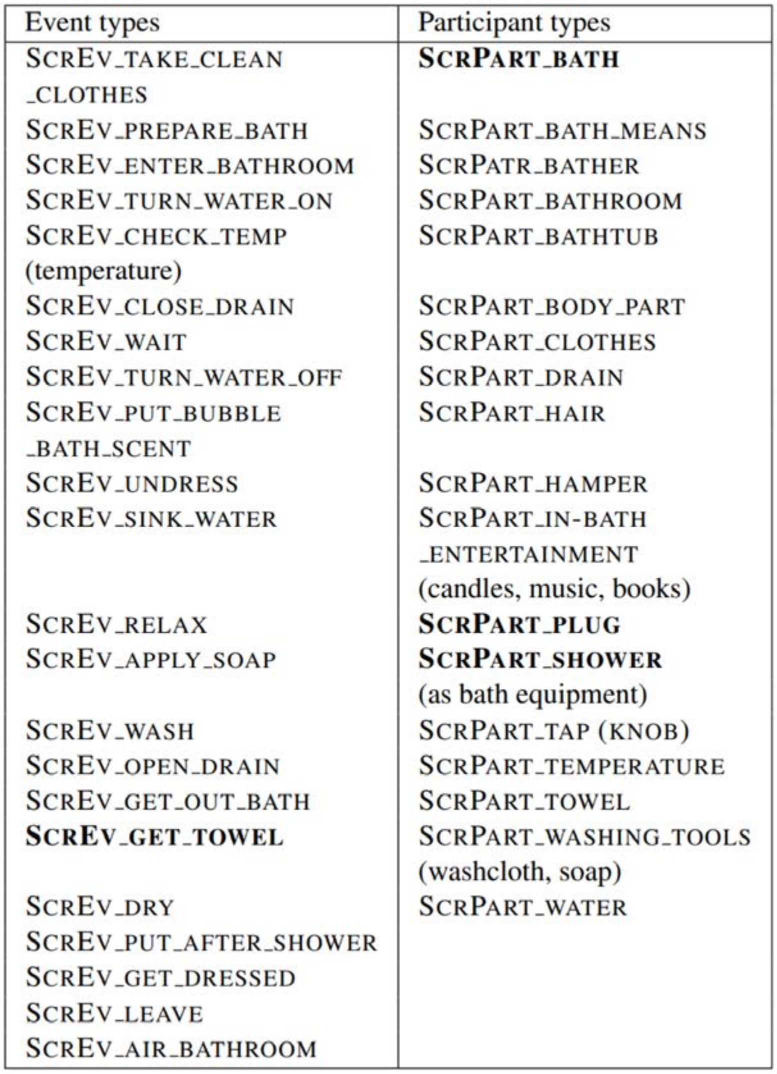
Scenario Name	#Stories	Avg. Sentences Per Story	Avg. Word Type Per Story	Avg. Word Count Per Story	Avg. Word Type Overlap
Riding in a public bus (Bus)	92	12.3 (4.1)	97.4 (23.3)	215.1 (69.7)	35.7 (7.5)
Baking a cake (Cake)	97	13.6 (4.7)	102.7 (23.7)	235.5 (78.5)	39.5 (8.1)
Taking a bath (Bath)	94	11.5 (2.6)	91.9 (13.1)	***197.5 (34.5)***	37.9 (6.3)
Going grocery shopping (Grocery)	95	13.1 (3.7)	102.9 (19.9)	228.3 (58.8)	38.6 (7.8)
Flying in an airplane (Flight)	86	**14.1 (5.6)**	**113.6 (30.9)**	**251.2 (99.1)**	**40.9 (10.3)**
Getting a haircut (Haircut)	88	13.3 (4.0)	100.6 (19.3)	227.2 (63.4)	39.0 (7.9)
Borrowing a book from the library (Library)	93	11.2 (2.5)	***88.0 (14.1)***	200.7 (43.5)	34.9 (5.5)
Going on a train (Train)	87	12.3 (3.4)	96.3 (19.2)	210.3 (57.0)	35.3 (6.9)
Repairing a flat bicycle tire (Bicycle)	87	11.4 (3.6)	88.9 (15.0)	203.0 (53.3)	33.8 (5.2)
Planting a tree (Tree)	91	***11.0 (3.6)***	93.3 (19.2)	201.5 (60.3)	***34.0 (6.6)***
Average	*91*	*12.4*	*97.6*	*216.9*	*37.0*

Figure 2.9: Statistics of InScript taken from the original paper.

are InScript-style stories that are relatively simple in language. In total, MCScript2.0 comprises 19,821 questions on 3,487 stories or on average 5.7 questions per story. These questions are supposed to ask about elements of script knowledge. The average length of the stories is 164.4 tokens; that of the questions and answers is 8.2 tokens and 3.4 tokens, respectively.

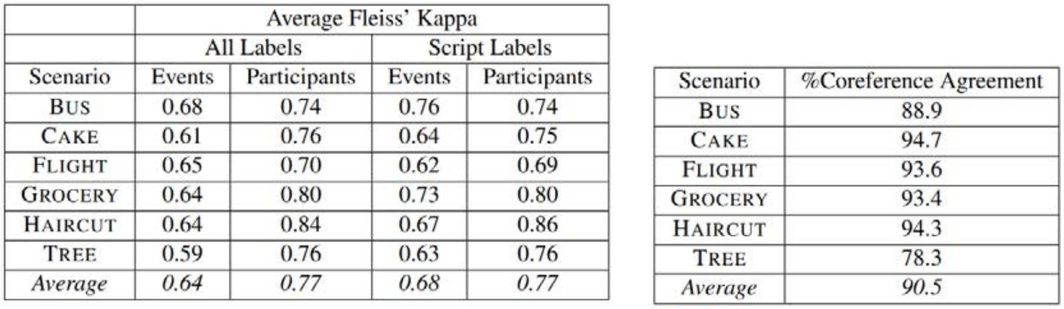
The questions are crowd-sourced. For each story, the authors take the fourth sentence as the *target sentence*. The workers are then instructed to focus on the target sentence and ask two types of special questions. Firstly, they should ask about a specific noun phrase in the sentence; these questions are supposed to be about script participants. Secondly. Workers should ask a temporal question about a verbal phrase in the target sentence. These questions are supposed to be about the temporal structure among script events. The target sentence is likely to contain the answer to the question to collect. When the questions are presented, it will be hidden from the text, to create a situation where the answer needs to be inferred according to script knowledge. In another round of crowd-sourcing, the workers are instructed to identify the quesitons that could be answered by checking the text or inferring from common sense. These questions are to be kept in the dataset.

The answers are also crowd-sourced. Workers are instructed to provide the correct answer and a plausible incorrect answer to each question. Each question is given to 5 workers; the final correct answer is decided by a majority vote (answers within a Levenstein distance of 3 are considered identical); the final incorrect answer is selected from those provided by the workers via *adversarial filtering*, where questions more difficult to a simple classifier are selected through the process. The result is a challenging dataset for machines, with then state-of-the-art reading comprehension



Event types	Participant types
ScrEv_take_clean_clothes	**ScrPart_bath**
ScrEv_prepare_bath	ScrPart_bath_means
ScrEv_enter_bathroom	ScrPatr_bather
ScrEv_turn_water_on	ScrPart_bathroom
ScrEv_check_temp (temperature)	ScrPart_bathtub
ScrEv_close_drain	ScrPart_body_part
ScrEv_wait	ScrPart_clothes
ScrEv_turn_water_off	ScrPart_drain
ScrEv_put_bubble_bath_scent	ScrPart_hair
ScrEv_undress	ScrPart_hamper
ScrEv_sink_water	ScrPart_in-bath_entertainment (candles, music, books)
ScrEv_relax	**ScrPart_plug**
ScrEv_apply_soap	**ScrPart_shower** (as bath equipment)
ScrEv_wash	ScrPart_tap (knob)
ScrEv_open_drain	ScrPart_temperature
ScrEv_get_out_bath	ScrPart_towel
ScrEv_get_towel	ScrPart_washing_tools (washcloth, soap)
ScrEv_dry	ScrPart_water
ScrEv_put_after_shower	
ScrEv_get_dressed	
ScrEv_leave	
ScrEv_air_bathroom	

Figure 2.10: The template for taking a bath taken from the original paper. The templates see some adjustments after the manual annotation, to add the items in boldface.



	Average Fleiss' Kappa			
	All Labels		Script Labels	
Scenario	Events	Participants	Events	Participants
BUS	0.68	0.74	0.76	0.74
CAKE	0.61	0.76	0.64	0.75
FLIGHT	0.65	0.70	0.62	0.69
GROCERY	0.64	0.80	0.73	0.80
HAIRCUT	0.64	0.84	0.67	0.86
TREE	0.59	0.76	0.63	0.76
Average	*0.64*	*0.77*	*0.68*	*0.77*

Scenario	%Coreference Agreement
BUS	88.9
CAKE	94.7
FLIGHT	93.6
GROCERY	93.4
HAIRCUT	94.3
TREE	78.3
Average	*90.5*

Figure 2.11: Interannotator Agreements. Left: events and participants; right: coreference. The coreference agreement is the percentage of coreferring pairs annotated by at least 3 annotators out of 4.

model performing at 72 points accuracy.

Script-based NLU is beyond the scope of this thesis. But MCScript is interesting to us as it provides parallel descriptions of script scenarios with simple language. We will use these texts to perform automatic script acquisition.

2.4 Script Knowledge for NLP Applications

In this section we briefly go through a number of results where script knowledge proves useful for NLP applications.

2.4.1 Referent Prediction

Script knowledge offers specifications on how the participants are involved: the protagonist is a participant in most events; each participant participates in script events in specific patterns; for example, a cake is put in to the over to bake, and later taken out. Rahman and Ng (2012) shows that script knowledge is useful against ambiguities in coreference resolution. The paper investigates complex coreference resolution cases that coreference resolution heuristics and linguistics constraints like gender agreement, number agreement, syntactic salience, etc. struggle to deliver results. Among these complex cases are the instances in the Winograd schema challenge(Levesque et al., 2012, see Figure 2.13 for an example).
Rahman and Ng (2012) investigate the features extracted from narrative event chains in a number of tasks. As is shown by their ablation study, including these

Story: *We put our ingredients together to make sure they were at the right temperature, preheated the oven, and pulled out the proper utensils. We then prepared the batter using eggs and some other materials we purchased and then poured them into a pan. After baking the cake in the oven for the time the recipe told us to, we then double checked to make sure it was done by pushing a knife into the center. We saw some crumbs sticking to the knife when we pulled it out so we knew it was ready to eat !*

Question: *When did they put the pan in the oven and bake it according to the instructions?*
✓ *after mixing the batter*
× *after eating the cake*

Question: *What did they put in the oven?*
✓ *the cake mix*
× *utensils*

Figure 2.12: An example Ostermann et al. (2019) which includes two questions based on a text about baking a cake.

features improves performance of their model by more than 4 percent accuracy.

Modi et al. (2017) explore the potential to improve the prediction of referents with explicit script knowledge. They construct a referent cloze test from the stories of InScript and approach the task with a log-linear model. They include two script-related features: first, *participant type fit*, which characterizes 'how well the participant type of a candidate fits a specific semantic role of the governing predicate'; secondly, *predicate schema*, which is about event pairs that tend to appear consecutively in the text. Including these features increased performance by more than 10 points of accuracy.

2.4.2 Discourse Classification

Lee et al. (2020) use explicit script knowledge for discourse classification. They learn representations of events through the construction of *narrative graphs*. The narrative graph is a refined version of narrative chains. Lee et al. (2020) extract events, or predicate-argument structures from a large body of text. Together with these predicate-argument structures, they also extract relations among them. These relations are mostly discourse relations between sentences that include predicates-

a. *Putin issued Edward Snowden Russian citizenship because* ***he*** *represents freedom fighters.*

b. *Putin issued Edward Snowden Russian citizenship because* ***he*** *protects freedom fighters.*

Figure 2.13: A Winograd-style complex coreference resolution instance. The syntactic trees of this pair of sentences are isomorphic. For this pair of sentences, the referent of *he* cannot be determined from shallow linguistic features. Yet patterns in predicate-argument pairings might help with such cases.

argument structures. With the events and relations among them, they train a graph convolutional neural network to predict missing edges of the graphs. The training produces vector representations of events.

The event embeddings are used for implicit discourse sense classification, a shared task of CoNLL 2016 (Xue et al., 2016). The authors concatenate the event embeddings to the ElMo representation of both discourse arguments to feed into a classifier. Including these embeddings saw a performance improvement of at least 2 points F1 over several competitive systems.

2.4.3 Story Generation

Various works on story generation see traces of script knowledge. Fan et al. (2018,0) generate stories in a hierarchical way. Given a title, a plausible chain of predicate-argument structures is generated to sketch the story. Later, the complete story is generated conditioned on the sketch. Martin et al. (2018) also takes a hierarchical approach. They treat an event as a sequence containing S, V, O, and potentially a modifier M. Their system involves several sequence-to-sequence models. An *event-to-event* model incrementally generates sequences of events; an *event-to-sentence* model generates almost natural language text that contains placeholders for participants from each event. Zhai et al. (2019b) generates script stories from an agenda, which is a plausible sequence of InScript events that sketches the story. The agenda itself is generated according to the temporal script graph of each scenario learned from the annotations in InScript. All these works take a hierarchical approach, where an intermediate *outline* is first generated to sketch the story. Script knowledge is an important part of the procedural knowledge on which the plausibility of these sketches are established.

2.4.4 Summary

Script knowledge benefits a variety of NLP applications. It is used in different forms. It could be patterns in event sequences or explicit script knowledge like InScript. It could be encoded in vector representations. Without scenario specification, script knowledge could be extracted on a large scale. Yet without scenario specification, the outcome is inevitably ignorant of scenarios and the hidden elements of script knowledge. The availability of scenario-specific script knowledge, however, limits to what InScript offers, i.e. 10 daily scenarios. Script knowledge would need to reach a wider coverage to deliver practical usefulness.

Chapter 3

Script Story Generation

3.1 Introduction

Automatic story generation is the task of automatically crafting stories with appropriate contents and language. These texts need to be both fluent and coherent. Figure 3.1 provides some examples of machine-generated stories.

Establishing coherence is the main technical challenge in text generation. The text needs to comprise fluent language that correctly conveys the communication purpose, such as reporting a piece of news, quirying for some specific information, etc. Although modern neural language models can easily produce sensible, random text pieces, establishing control over the content of these texts is a different task. In general, the agent needs to master basic linguistic skills to produce cohesive texts; whereas selecting appropriate content to convey a specific information could mean bridging surface language and the physical world, which is extremely tough.

In terms of story generation, the coherence of the text is established by appropriately realizing a specific story line. Traditionally, story generation is performed with symbolic planning systems. Typically, a symbolic natural language generation architecture consists of three stages: text planning, sentence planning, and surface realization (see Figure 3.2). Text planning accounts for the highest level of planning, including discourse planning (grouping concepts and specifying discourse relations to arrange them) and content determination (plan sentences to express the concepts). It basically decides what each sentence should say. Sentence planning and surface realization would actually generate sentences to fulfill the plan. Sentence planning sets up the arena for surface realization according to the outcome of text planning. It decides the coreference relations in the sentence in question, as well as its word choices. Surface realization directly generates natural language text accordingly. It could be either a rule-based system, or a probabilistic model (see, e.g., Meehan, 1976; Riedl and Young, 2010; Busemann and Horacek, 1998).

Traditional story generation systems are capable of producing impressive, coherent stories, these systems rely heavily on manual knowledge engineering to select actions, characters, etc., properly, therefore generalizing poorly to unseen domains.

Early neural language models, such as those based on LSTM, exhibit impressive capabilities to generate fluent utterances, which drove the development of neural text generation systems (see, e.g., Mei et al., 2016; Wen et al., 2015). Yet, these models struggle to keep track of the generation progress or dialog history, and the coherence of the generated text deteriorates as the generation continues. The following few years saw the establishment of guided text generation, where generation

a. *A young girl named Sarah lived in a small village at the edge of a dense forest. She was known for her kindness and her love of animals. One day, while exploring the forest, Sarah stumbled upon a clearing and found a magnificent unicorn. The unicorn was injured and in need of help, so Sarah offered to take care of it.*

She brought the unicorn back to her home and nursed it back to health. In return, the unicorn granted Sarah three wishes. She wished for the forest to be protected, for the village to prosper, and for peace to reign throughout the land. The unicorn's magic worked, and the village flourished.

Years passed, and Sarah grew old. The unicorn returned to the forest, but Sarah never forgot the magical bond they shared. She passed down the story of the unicorn to her children, who in turn passed it down to their children. And so, the story of the unicorn and the girl who saved it lived on, bringing hope and joy to all who heard it.

b. *Once upon a time, a family decided to take a trip by train. They went to the train station and saw a big and shiny train waiting for them. The train was long and colorful, with lots of windows for them to look out of.*

The family found their seats and settled in. They were so excited to be on a train! As they started moving, they could feel the gentle swaying of the train. They watched as the scenery outside changed, passing by fields, forests, and even mountains. They saw other trains going by and waved at the people on those trains.

The train made stops at different stations, and new passengers got on and off. The family was given a treat, like a snack or a drink, and they enjoyed it as they continued their journey.

After some time, they arrived at their destination. They got off the train and felt grateful for their fun and memorable trip. They can't wait to take the train again!

Figure 3.1: Automatically generated stories by ChatGPT (`https://chat.openai.com/chat`). In particular, **b.** is a script story on taking a train.

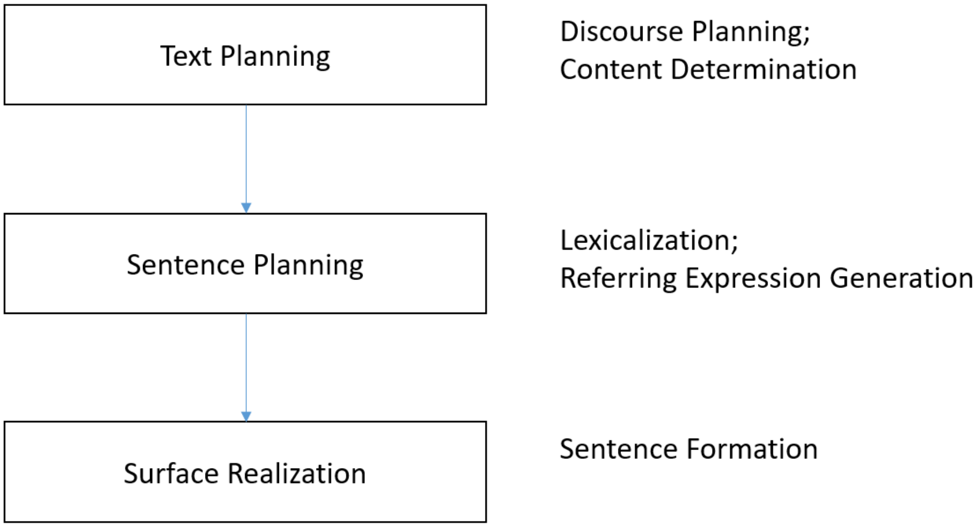
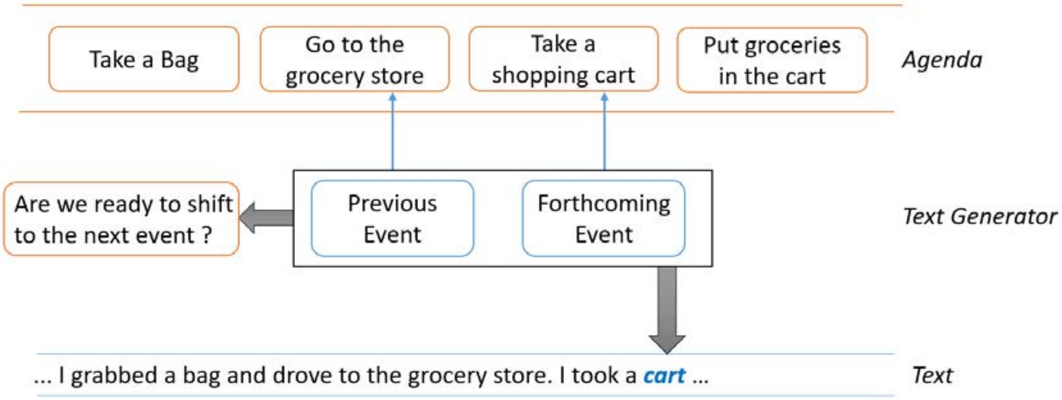


Figure 3.2: A symbolic pipeline of text generation. Planning and generation go from higher to lower levels

systems of dialog, story, sports review, etc. include explicit planning components to keep track of the generation progress and inform the surface realization component. Generation could be planned with a list of important entities (Kiddon et al., 2016), a sequence of predicates (Fan et al., 2019), etc.

This chapter is focused on the generation of script stories, i.e. narratives of script scenarios, like taking a train. These stories are supposed to be about typical experiences of the scenario. Figure 3.1 **b.** gives an example. Script story generation is based on corpora like InScript, which contains annotation of script events and participants.

Earlier work (Zhai et al., 2019a, ; Section 3.2) has developed methods to establish global coherence in script story generation, such that the generated stories describe the procedure of performing a script scenario in a plausible manner. For example, a story on taking a train could be generated based on an agenda like *buy_ticket* → *go_to_train_station* → *board_train* → *train_arrives* → *exit_train*. Their model features the combination of a symbolic component and a neutral component. The symbolic component learns from the annotations in InScript to produce an agenda, a sequence of events that sketches a realistic instantiation of the scenario. The agenda is then fed to the neural component to condition the generation of the story. The surface realization component verbalizes the agenda in linear order, event-by-event. See Figure 3.3 for an illustration.

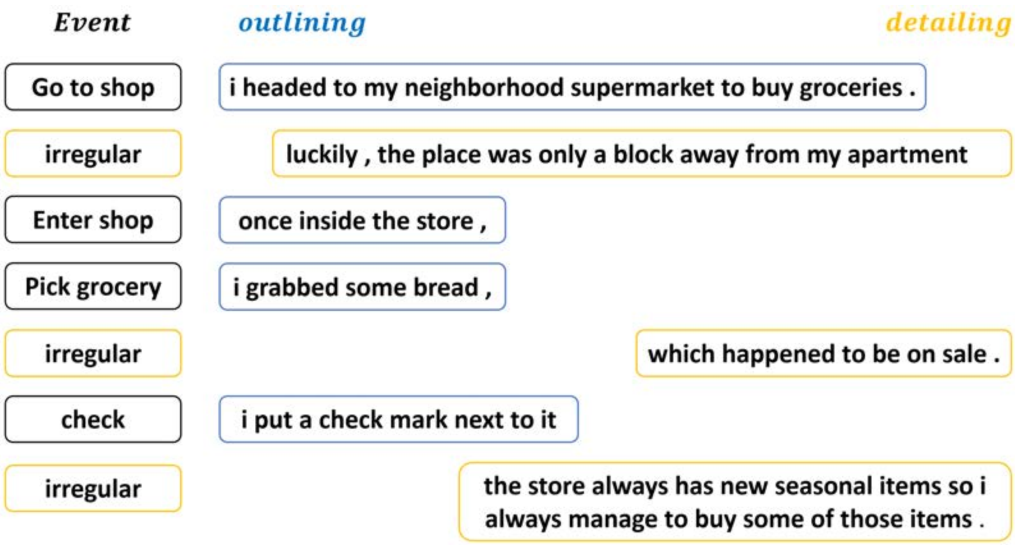


yesterday i went grocery shopping . i made a list of my list and drove to the grocery store . when i entered the store , i grabbed a shopping cart and pushed the cart down to the meat aisle . i got all my items , and crossed items on my list . i went to the checkout register and paid for my groceries . i put my groceries in my cart and left .

Figure 3.3: An illustration of the model architecture of Zhai et al. (2019a) and a story it produces. Based on the agenda generated by the planner, its surface realization module produces two outputs: the prediction of the successive word and an indicator predicting whether the generation should move to the next event. The stories it generates are coherent but very short.

Zhai et al. (2019a) Generates coherent stories that describes the process of their scenarios soundly. Script knowledge was a most crucial ingredient in the process, as it guarantees the plausibility of the agenda and enables the generation model to track how much of the agenda has been realized in surface text.

In Section 3.3 we exploit script knowledge for an additional functionality in story generation, that is, including relevant details in the generated stories. The stories generated by Zhai et al. (2019a) are coherent but very short and barely contains interesting contents. We seek to improve the generation system to be more informative on the specific instantiation of the script scenario. Selecting these contents to be generated appropriately is not trivial. The model we propose is based on the observation that story generation could be seen as the joint effort of two components: one develops the story line, whereas the other provides details about the current situation without explicitly progressing with the story line. These components are approached with different model components. See Figure 3.4 for an illustration of the idea and a story generated by the model.



i went grocery shopping today . i made a list of things that i needed before i left my house . i looked at the grocery store immediately , i drove to the grocery store that is always empty . i parked the car , walked inside and grabbed a shopping cart . i proceeded down the produce aisle , which happened to be on sale . i went to the back and chose my favorite peanut butter . i went to the cashier , which was a short line . i had to wait in line since there were some people ahead of me . i put my items on the conveyor belt and then she scanned my items . i said hi to the cashier , i paid my bill , and another man placed all my items in my shopping bags . i made sure to put the cart away , too . once i had everything i needed , i then picked up my bags and left the store . on the way out i decided to rent a redbox movie for the night since the kiosk was right by .

Figure 3.4: An illustration of the architecture of the model that includes rich details in its generation, and a story it produces. The left column lists the agenda events and, on the right, the text segments corresponding to each event. The craft of the story naturally decomposes into an alternating effort between ***outlining*** and ***detailing***. The story packs more content than those generated by Zhai et al. (2019a) and arranges these contents appropriately.

As Zhai et al. (2019b), our model receives an agenda as the sketch of the story to generate. The agenda, however, contains irregular items which indicate extra details are required instead of developing the story line. We train different sequence to sequence models to develop the story line and to supply additional details. Script knowledge, which conditions the generation, is the key element for selecting appropriate contents that are consistent with the scenario and the ongoing story generation.

The works presented in this chapter exhibit the usefulness of script knowledge to the application of story generation. Script knowledge allows the generation systems to establish global coherence, as well as generating more details locally. Yet, the limitation of current script resources is also evident from these works, that is, the coverage of exhisting resources. The models in this chapter train on finely annotated data, which is available only for a few scenarios. To be of practical usefulness, its training data need to reach a reasonable coverage of the script scenarios one encounters in daily life. In the following chapters, we develop methods to acquire script knowledge automatically from surface text, which makes it affordable to extend the coverage of script knowledge. In chapter 6, we revisit the task of script story generation, where we experiment with automatically acquired script knowledge.

3.2 Script Knowledge in Globally Coherent Story Generation

Script knowledge packs information about how daily activities are usually carried out. It specifies the motivations (for fixing a flat tire, for taking a taxi or a train, etc.), the participants, the locations, the necessary steps, etc. With knowledge of these aspects, we are able to sketch a narrative of the scenario; we could judge whether a narrative of a scenario is plausible: did the passenger pay for his train ride? Did he exit the train after it stops? One possible application is to establish global coherence in narrative texts about these scenarios. This is realized by Zhai et al. (2019b), which we introduce in this section.

3.2.1 Method

Overview

Zhai et al. (2019b) is based on InScript, where event annotations are available. The authors condition surface realization on an *agenda*, which is a plausible sequence of events that instantiates a scenario of interest. The agenda could be acquired from script knowledge, and in the meanwhile, it serves as a sketch of the story to generate. Their model consists of three modules. Firstly, a symbolic ***agenda generator*** that produces an agenda given a specific scenario (e.g., baking a cake). Second, a neural ***surface realization module***, which verbalizes the agenda event by event.

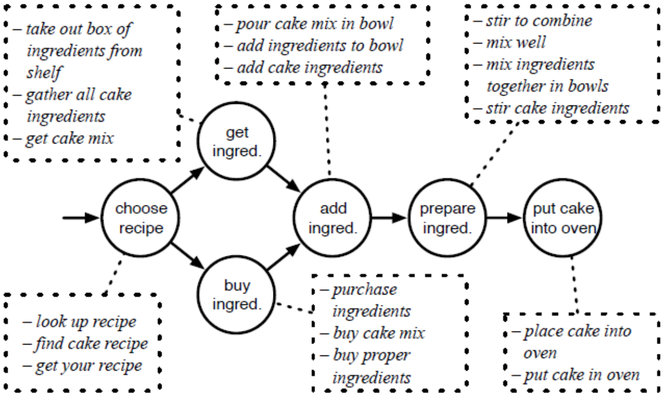


Figure 3.5: The *Temporal Script Graphs* for the BAKING A CAKE scenario induced from InScript, taken from (Wanzare et al., 2017). The nodes are the event clusters, whereas the dashed boxes include some possible utterances that correspond to these clusters.

It performs two tasks. (1) It behaves like a language model, which predicts the next word of the story given the current event and the progress of generation. (2) It determines whether the current event has been completely realized so that the generation could move to the next event on the agenda. Finally, a ***story generator*** that assembles the whole pipeline as follows. (1) Call the *agenda generator* to generate an agenda. (2) Create a *seed*, a short, plausible story beginning, to initialize surface realization, e.g., '*yesterday i went grocery shopping*' for a grocery shopping story. (3) Iteratively call the surface realization module to perform a beam search (see, e.g., Sutskever et al., 2014) and generate a complete story. (4) Post processing. Remove occasional excessive repetitions in the generated story. More precisely, when a word or phrase is repeated at least three times, the third repetition is deleted. e.g., 'i like the tree very very very much' becomes 'i like the tree very very much'. The generation terminates when the agenda is exhausted and a sentence-terminating punctuation is generated.

The Agenda Generator

Given a scenario, the *agenda generator* goes through its temporal script graph and samples a path through it. For the example given in Figure 3.5, the path would start out with "choose recipe" and continue with either "get ingredients" or "buy ingredients", followed by "add ingredients", until the end of the graph is reached. The sampling is based on an event n-gram language model trained on the training set.

The agenda generator also decides whether each event should be realized. Naturally generated stories rarely mention all of the events, and this component enables the model to mimic this behavior: the probability of event realization depends on the likelihood of the event given its predecessor $p(e|e')$, which is estimated on the training data using an event bigram model. To avoid excessive discontinuity in the realization, the *agenda generator* is prohibited to skip two consecutive events. The outcome of this process is an agenda, a plausible sequence of events.

The Surface Realization Module

Their surface realization module is a GRU (Cho et al., 2014) language model, modified to enable two additional functionalities. (1) We condition the prediction of the successive word on the generation progress. (2) It determines whether the current event has been fully verbalized. If so, the surface realization module shifts its focus one event onward along the agenda and begins to verbalize the next event. See Figure 3.6 for a conceptual illustration.

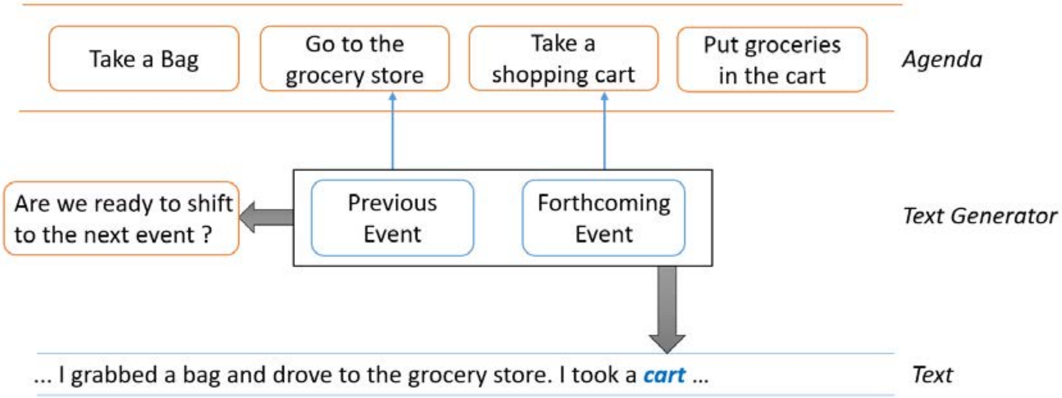


Figure 3.6: An illustration of the surface realization module. It produces two outputs: a distribution over the vocabulary that predicts the successive word and a boolean-valued variable that indicates whether the generation should move to the next event.

For the first functionality (see Figure 3.7 for the model architecture), Zhai et al. (2019b) condition the prediction of the next word on both the previously verbalized event (the ***preceding event***) and the event that should now be verbalized (the ***forthcoming event***). Intuitively, the surface realization module will be informed with something like '*I have taken a shopping cart, now tell me how to get my groceries*'. The authors train a dense vector representation for each event in the corpus, which is termed an ***event vector***. Generation is conditioned on event vectors e_t^p of the *preceding* event and e_t^f of the *forthcoming* event:

$$d_t = Softmax(D[o_t; e_t^p; e_t^f])$$

here d_t is the output distribution that predicts the successive word; D is a linear layer; ';' stands for vector concatenation; $o_t = Wh_t$ is the content of the GRU language model where h_t is the GRU cell states and W is another linear layer. To further relate surface realization with the progress of generation, we concatenate the event vectors with the embedding of the previous word as input to the GRUs:

$$h_t = GRU([x_{t-1}; e_{t-1}^p; e_{t-1}^f])$$

The word embeddings are initialized with Google's pre-trained word2vec vectors[1] (see, e.g., Mikolov et al., 2013). A binary classifier determines whether the *forthcoming event* has been instantiated, i.e., whether the model is ready to move onward:

$$a_t = Softmax(A[h_t; e_t^p; e_t^f])$$

here A is a projection matrix; a_t is a 2-dimensional vector. If $a^1 > a^0$, the surface realization module decides that *forthcoming event* has been completely narrated and should move one event onward to continue the generation; otherwise, it sticks with the current *forthcoming event* to complete its instantiation.

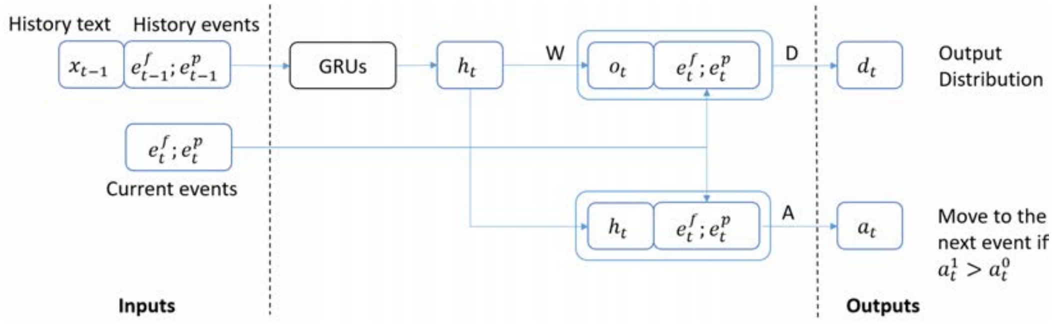


Figure 3.7: An illustration of the surface realization module. It outputs the distribution of the next token d_t, as well as a_t, which determines whether to proceed to the next event.

As there are more tokens than events in the corpus, the aforementioned binary classification is biased. The ratio between the categories is about 17 : 3. To improve recall on event-shift decisions, they use a weighted cross-entropy loss on output a_t:

$$L_a(a_t, a_t^*; \gamma) = \begin{cases} -(1-\gamma)\log a_t^0, & a_t^* = (1,0) \\ -\gamma \cdot \log a_t^1, & a_t^* = (0,1) \end{cases}$$

---

[1] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit

here a_t^* is the ground truth; the weight coefficient γ is a hyperparameter. The final loss function is

$$L(a_t, a_t^*, d_t, d_t^*; \gamma, \beta) = H(d_t, d_t^*) + \beta \cdot L_a(a_t, a_t^*; \gamma)$$

where $H(\cdot, \cdot)$ denotes the cross entropy between the parameters; d_t^* is the ground truth distribution of the next word; β is another hyperparameter.

3.2.2 Evaluation

The following generation systems are included in the evaluation.

Human Author
Stories directly taken from InScript. Expected to produce an upper bound of the evaluation.

Full
The model described in Section 3.2.1.

GRU
A plain GRU language model trained on InScript, intended to be a baseline that has no specific global coherence control. Its generations are seeded with scenario-specific beginnings for some relevance. For example, the seed for Going Grocery Shopping is '*yesterday i went grocery shopping .*'

Neural Checklist Reimplementation of the *neural checklist* model by Kiddon et al. (2016), briefly post-processed as described in Section 3.2.1.

Random Event Order
The ablated model without the *agenda generator*. As a substitution, the agenda now consists of mere i.i.d. samples of events.

For some intuition, see Table 3.1 for sample generations from these systems.

Evaluation Method

Zhai et al. (2019b) performs a human evaluation through crowd-sourcing, to evaluate the following aspects of generated stories.

Syntax
The syntactical correctness of sentences.

GRU

yesterday i went grocery shopping . i did n't know anything all the items in my cart , so that it was ready to pick up up . i got my items off , and found it to the front of the store . i was on narita shopping cart because i had less than twenty of them . i grabbed my cart and went to the cashier . i paid for the items , then i gave the cashier my money and my bag , checked my bags up , and the other items in my hand bag . i am glad i 'm flying for my plane and go through security .

Neural Checklist

yesterday i wanted to go grocery shopping . when i went to the front of the store , i went to the grocery store and drove to the front of the store . i went to the cashier and drove to the front desk .

Random Event Order

yesterday i went grocery shopping . i grabbed a shopping cart and put the items on the list . i put my items in the cart and drove to the store . i drove to the grocery store and went to the grocery store . i made a list of all the items i would need . i put my items on the cart and headed to the store . when i entered the store , i grabbed a shopping cart and walked inside . i put the cart away in the cart .

Agenda: *evoking→ take shop cart→ put conveyor→ get groceries→ go grocery→ move section→ make list→ check off→ put conveyor→ leave→ enter→ take shop cart→ enter→ return shop cart→ story ends*

Full

yesterday i went grocery shopping . i made a list of my list and drove to the grocery store . when i entered the store , i grabbed a shopping cart and pushed the cart down to the meat aisle . i got all my items , and crossed items on my list . i went to the checkout register and paid for my groceries . i put my groceries in my cart and left .

Agenda: *evoking→make list→go to store→enter→take cart→move along sections→take grocery→check list→go to checkout→pay→pack grocery→leave→ story ends*

Human Author

yesterday i went grocery shopping . i took my grocery list with me , along with some reusable shopping bags . my grocery list has all the items i want to buy on it . i selected a shopping cart from in front of the store , and went inside . i put my reusable bags in the cart . i looked at my list and started in the produce section . i put different vegetables and fruits into my cart . next i wheeled my cart to the cereal aisle and took a box of cereal . i went through the store aisle by aisle and selected my groceries . each aisle is organized by types of food and non-food items . one aisle has dried pasta , canned tomatoes , rice , and sauce . i selected a few boxes of pasta and some rice . another aisle carries plastic wrap , trash bags , and aluminum foil . as i went through the store , i kept looking at my list to see what i needed next . when i added each item to my cart , i crossed it off my list . my last stop was the dairy aisle where i got milk and eggs . when i had all the groceries i wanted , i went to the cash register ans stood in line . when it was my turn , i put each item on the conveyor belt and the cashier scanned each one . a bagger put all of the

	Agenda Coverage**	Syntax	Inclusion	Order	Relevance
human author	86%	0.86	0.91	0.93	0.83
full	**71%**	**0.75**	**0.67**	**0.75**	**0.88**
random event order	50%	0.45	0.46	0.14*	0.71
Neural Checklist	20%	0.54	0.34	0.27	0.53
GRU	n/a	0.33	0.24	0.11*	0.22

*: difference between the pair is not statistically significant due to paired T-test on a significant level $\alpha = 0.05$.

**: answers to the agenda coverage questions yield a Fleiss' kappa of 0.34.

Table 3.2: Results from human evaluation. The highest scores from the automatic systems are displayed in bold.

Human Author : event *'make a shopping list'* in scenario *'going grocery shopping'*
... next , i used the wipes the store provides at the entrance and wipe off the handle of the shopping cart , and my hands , so i know my hands will stay clean while i choose my food . then ***i took out the shopping list i wrote at home*** and i started . i always start with heavy things ...
Full Model : event *'place fertilizer'* in scenario *'planting a tree'*
... yesterday i planted a tree. first , i decided to buy a small apple tree . i got a shovel and drove to the home . i found a perfect spot in my backyard and dug a hole . ***i put*** the soil in the hole and then watered it . ...

Table 3.3: Examples where the instantiations of agenda items did not get approved by the evaluators. Up: event instantiation that was not explicit enough; down: event that was not instantiated due to an error in the output a of the surface realization module.

> **Global Coherence**
> The global coherence of a story with regard to the given scenario, e.g. going grocery shopping. We evaluate from three aspects: ***Inclusion*** (does the story cover the most necessary steps about the topic?), ***Relevance*** (does the story stay on-topic and rarely mention anything irrelevant to the topic?), and ***Order*** (does the story describe the activities relevant to the topic in a plausible order?)
>
> **Agenda Coverage**
> The correspondence between the generated story and the agenda with which it was fed. The participants were asked if each of the items of the agenda has been realized in the story.

Participants receive five questions per story: for *Agenda Coverage*, participants were asked to check the items on the agenda that were mentioned in the story they saw; for the other four aspects, participants were asked to rate on a scale of 1 to 4. The evaluation experiment was implemented with Lingoturk(Pusse et al., 2016); the experiments were conducted through Prolific[2].

Results

Table 3.2 illustrates the results of human evaluation. The GRU model, a plain language model without coherence modeling, yields the worst performance on all metrics. The output wildly changes between topics and is incoherent globally; the poor coherence probably also negatively affects human judgments on syntactic correctness. The neural checklist showed better performance than plain GRUs, but did not include the most necessary steps of the scenario. It seems that the model cannot correctly track the progress of the generation, which might be a consequence of the limited amount of training data, because its attention-based content planning component cannot make use of the order information and has to learn it from the data, thus having a substantially higher demand on training data.

The Full model was able to significantly outperform all other automatic variants and received positive scores for all criteria. It reflects well the events on the agenda and usually includes the most necessary steps of the scripts in a plausible order, which indicates decent global coherence. It even received a higher *relevance* score than Human Author. However, this may result from its often producing shorter stories than the human originals. Its *agenda coverage* score is lower than that of Human Author. Two sources are responsible for these errors: (1) event instantiations

---

[2] `https://prolific.ac/`

are sometimes not recognized as such by participants because they are not explicit enough; this is also the reason why the agenda coverage score for the original human texts is less than 100%. (2) Errors in the event termination judgments of the surface realization module: when the surface realization module wrongly decided that the *forthcoming event* has been instantiated, it would simply skip the event in the generation.

Random Event Order witnessed a dramatic performance drop compared to Full. Its order score is not significantly different from that of the GRU baseline. That means, the *agenda generator* was capable of performing reliable text planning and incorporating global coherence. It retained a high relevance score (i.e., it still stays on topic), as the agendas it uses are still about the respective scenarios.

3.2.3 Summary

In this section, we introduced the story generation model proposed by Zhai et al. (2019b) which establishes global coherence with script knowledge. The key design is to sketch the story with an agenda and verbalize the events one by one, conditioned on the progress. With a finely annotated corpus like InScript, the model learns the alignment between events and surface text and generates coherent stories with appropriate contents.

3.3 Script Knowledge in Story Generation with Rich Details

3.3.1 Motivation

Zhai et al. (2019b) condition text generation on a plausible sequence of events that accomplishes the scenario of interest. The generated stories correctly describe how the scenario is instantiated. Yet, they hardly go beyond that and end up boring. As an example, Figure 3.8 shows a story about grocery shopping. The story is globally coherent as a narrative of grocery shopping, but it is hard to imagine that anyone is attracted by it: it hardly provides any further information than what the topic grocery shopping already indicates.

Stories should be interesting, in addition to being globally coherent. That is to say, apart from realizing a story line, the text should include more details about the situation: was it an empty fridge that made it necessary to go grocery shopping or was I simply bored? Maybe the vegetables were not fresh; maybe I encountered *yesterday i went grocery shopping . i made a list of my list and drove to the grocery store . when i entered the store , i grabbed a shopping cart and pushed the cart down to the meat aisle . i got all my items , and crossed items on my list . i went to the checkout register and paid for my groceries . i put my groceries in my cart and left .*

Figure 3.8: A story about grocery shopping generated by Zhai et al. (2019b), which is globally coherent but not engaging.

people demonstrating during G7 and had to wait 30 minutes at a cross. Maybe I met an old friend along the way or lost my purse at the grocery store. It is such a variety that makes a narrative of going grocery shopping somewhat interesting.

In this section, we propose a story generation system that takes advantage of script knowledge to include rich details. To this end, we take a step further in the exploitation of script knowledge, alongside establishing global coherence. Knowing how a scenario typically instantiates not only allows us to sketch a specific story in a realistic manner; it also allows us to judge what and when additional details are appropriate. It is plausible to go grocery shopping or go to the cinema when I am bored, but it does not make much sense if I decide to fix a flat tire when bored. Similarly, figuring out that vegetables are fresh is only meaningful before queuing up to pay at the grocery store. With script knowledge, we would be informed of the process of the scenario and make such judgements.

We regard the process of story writing as a joint effort of two different components, suggesting that a story generation system would benefit from modeling them differently. Components are (1) ***Outlining***, where a story realizes a (usually) linear story line step by step, thus establishing the global coherence of the text. (2) ***Detailing***, where the author gives details about some of the steps in the story line so that the story becomes informative and interests its reader. A neural story generation system that conditions the generation on the story line achieves the former, but may well fail the latter.

We present a model that features two different generation components: (1) an ***outliner***, which proceeds the main story line to establish global coherence; and (2) a ***detailer***, which provides relevant details to the story in a locally coherent manner. The input of the model will still be an agenda. The model generates a story according to the agenda, segment by segment, each segment realizing one event in the agenda. An illustration is given in Figure 3.4.

3.3.2 Data and Pre-processing

The experiments are based on InScript. Recall that our model generates in a segment-by-segment manner, each of which realizes an event. To this end, we need to correspond each event to the segment of surface text that realizes it. Therefore, as a preprocessing step, we automatically dissect the stories in InScript to assign each event annotation to its corresponding text ***segment***. Our segmentation process prioritizes assigning each event label a text segment that retains relatively complete syntactic structure. This means that a complete SVO structure is preferred over a mere verbal phrase. The segmentation is rule-based and mainly relies on part-of-speech tags; the process is defined in Figure 3.9.

We use Spacy(Honnibal and Montani, 2017) for the POS tagging. It is trivial to prove that the heuristics in Figure 3.9 are well defined for our purpose, i.e. the output sequence of events and sequence of segments are of the same length; the segments do not overlap and sum up to the entire story. It is an engineering solution; yet, it yields segments of decent quality. See Table 3.4 for some examples.

3.3.3 Model

Overall, our model takes the sequence-to-sequence format. It has one encoder that tracks the history of the generation; meanwhile, it has two different decoders that correspond to the two different components of story crafting: an ***outliner***, which generates regular segments to instantiate the regular event items on the agenda; and a ***detailer***, which generates irregular segments to provide details to the story. The model generates a story segment-by-segment, alternating between the decoders to address one event at a time. Therefore, the neural part of the model receives as input (1) the input sequence, i.e., segments $s = \langle s_1 \dots s_{i-1} \rangle$ that are already generated and (2) the agenda $a = \langle e_1 \dots e_n \rangle$ that consists of n events in total; as output, it generates the next text segment s_i that corresponds to the current event e_i, with the respective decoder.

Encoder

We encode the input sequence in an agenda-aware manner. Let $l(i)$ be the length of the history, that is, the number of tokens in the segments $s = \langle s_1 \dots s_{i-1} \rangle$. For the representation of the input sequence at the i-th step, we expand the embeddings of its tokens with that of the event it realizes:

$$emb(i) = (\varphi_w(t_j); \varphi_e(e_i))_{j \leq len(s_i)}$$

event	text segment
evoking	yesterday i went grocery shopping .
take bags	i took my grocery list with me , along with some reusable shopping bags .
get groceries	my grocery list has all the items i want to buy on it .
take shop cart	i selected a shopping cart from in front of the store ,
enter	and went inside .
reln	i put my reusable bags in the cart .
check list	i looked at my list and started in the produce section .
get groceries	i put different vegetables and fruits into my cart .
move section	next i wheeled my cart to the cereal aisle
get groceries	and took a box of cereal .
move section	i went through the store aisle by aisle
get groceries	and selected my groceries .
unrelev	each aisle is organized by types of food and non-food items .
unrelev	one aisle has dried pasta , canned tomatoes , rice , and sauce .
get groceries	i selected a few boxes of pasta and some rice .
unrelev	another aisle carries plastic wrap , trash bags , and aluminum foil .
move section	as i went through the store ,
check list	i kept looking at my list to see what i needed next .
get groceries	when i added each item to my cart ,
check off	i crossed it off my list .
unrelev	my last stop was the dairy aisle
get groceries	where i got milk and eggs .
unrelev	when i had all the groceries i wanted ,
wait	i went to the cash register ans stood in line .
unrelev	when it was my turn ,
put conveyor	i put each item on the conveyor belt
cashier scan/weight	and the cashier scanned each one .
pack groceries	a bagger put all of the groceries into my reusable bags .
pay	i paid ,
get receipt	and then the cashier gave me a receipt .
bring vehicle	i loaded the bags of groceries into the trunk of my car
leave	and drove home .

Table 3.4: The segmentation of a story on going grocery shopping according to the rules in Figure 3.9.

Input All event annotations $e = e_1 \cdots e_n$ in their original order. Text segments $s_1 \cdots s_{n-1}$ where for all i, s_i collect tokens from the one that receives annotation e_i to the one before the token that receives e_{i+1}.

Output

1. $e'_i \cdots e'_m$, a sub-sequence of e.
2. Text segments $s' = s'_1 \cdots s'_m$, where for all $i \leq m$, s'_i realize the event e'_i.

Rules We say event label e_1 ***dominants*** event label e_2, noted as $e_1 >_d e_2$, if

e_1 is regular, and e_2 is not, or

e_1 is *RelNScr* (relavant non-script event) and e_2 is *Unrel* (unrelated).

For all s_i,

1. If s_i includes one punctuation in ('.', '?', '!', ';'): mark the token as 'split'.
2. Otherwise, if $e_i = e_{i+1}$ or ($e_i >_d e_{i+1}$ and $RelNScr >_d e_{i+1}$), mark s_i as 'merge'.
3. Otherwise, if s_i contains a comma, mark the comma as 'split'.
4. Otherwise, if the length of s_i is less than 5, mark s_i as 'merge'.
5. Otherwise, if s_i contains an 'and' which receives a 'CCONJ' from the POS tagger, or a 'but' which receives a 'CCONJ', mark the token as 'split'.
6. Otherwise, if s_i contains any token that receives an 'ADP', mark it as 'split'.
7. Otherwise, mark s_i as 'merge'.

For all s_i, if s_i is marked 'merge', mark a $>_d -$minimal element in e_i, e_{i+1} as 'del'.

Finally, split the entire story before each token marked 'split' to acquire s'; collect the items in e not marked as 'del' to acquire e'.

Figure 3.9: The rules that segments each story, assigning one segment to each event label. When event annotations are too dense, less interesting events are omitted to create segments with a closer to complete syntactic structure.

Here $\varphi_w(\cdot)$ denotes the word embeddings, which we initialize with pre-trained Glove embeddings (Pennington et al., 2014). $\varphi_e(\cdot)$ is the event embeddings; these embeddings are initialized randomly. $emb(i)$ is then fed to the encoder:

$$\boldsymbol{f_i} = Enc(emb(i)) \tag{3.1}$$

Here $Enc(\cdot)$ is a single layer Bi-LSTM sequence encoder. The output $\boldsymbol{f_i}$ is a sequence of vectors of the same length as $emb(i)$.

Outliner

The *outliner* generates the text segment corresponding to the next event e_i on the agenda, if it is regular. Here, we use a Bi-LSTM sequence decoder that is augmented by dot product attention (Luong et al., 2015) over the encoded sequence $\boldsymbol{f_i}$. Dec_0 denotes the decoder function at a specific time step. During the generation of a text segment s_i, at each decoding step t, the outliner produces a distribution of the next token over the vocabulary:

$$p_t = Dec_0(\varphi_w(token_{t-1}); att(d_{t-1}, \boldsymbol{f_i}); \varphi_e(e_i)) \tag{3.2}$$

Here, d_{t-1} is the inner state of the decoder before generating the t-th token of the current segment. $att(\cdot, \cdot)$ denotes dot-product attention, i.e.

$$\begin{aligned} \forall j < |\boldsymbol{f_i}| : \alpha_i &\leftarrow \langle d_{t-1}, \boldsymbol{f_i}_j \rangle \\ att(d_{t-1}, \boldsymbol{f_i}) &\leftarrow \sum_{j<|\boldsymbol{f_i}|} \alpha_j \boldsymbol{f_i}_j / \sum_{j<|\boldsymbol{f_i}|} \alpha_j \end{aligned}$$

where $|\boldsymbol{f_i}|$ denotes the length of $\boldsymbol{f_i}$. Thus $att(d_{t-1}, \boldsymbol{f_i})$ attentively collects information from the encoder. $token_{t-1}$ is the token generated in the previous time step $t-1$; $\varphi_e(e_i)$ denotes the embedding of the target event. During the inference phase, the complete segment will be generated with beam search.

Detailer

If the next event e_i is irregular, the *detailer* generate a text segment for it. Whereas *outliner* gets a regular event e_i like *pick up groceries* that is informative of the content of the next segment, *detailer* only gets an irregular event type that is the same for the entire scenario. An irregular event offers little information, and the generation tends to be generic. This is the main technical challenge of the model; we tackle this issue with the following efforts.

(1) conditioning. To specify the content of the current segment, we condition the decoding process on its most important context: the previous regular event e_- and the successive regular event e_+. Thus, in each decoding step t, the distribution of the next token is given by:

$$q_t = Dec_0(\varphi_w(token_{t-1}); att(d_{t-1}, \boldsymbol{f_i}); \varphi_e(e_-); \varphi_e(e_+)) \tag{3.3}$$

Here, $Dec_0(\cdot)$ is once again a single layer Bi-LSTM step.

(2) extra regularization. The *detailer* also adopts the maximum mutual information (MMI) objective. It is first used in conjunction with `seq2seq` models by Li et al. (2016), to promote the generation of specific, meaningful texts. It moderately suppresses the generation of generic text. The idea is that instead of maximizing the likelihood of the data, one could maximize the mutual information $I(c, s)$ between the context c and the generation s, which promotes the correspondence between the two. As a result, the generation will be more specific to its context and thus, we hope, more informative. The entire MMI decoding objective generalizes to:

$$s^* = \arg\max_s [log(p(s|c)) - \lambda \cdot log(p(s))] \tag{3.4}$$

which is the maximum likelihood along with a negative language model term, so it is also termed an *anti-LM* objective. In practice, equivalently, we follow the approach proposed by Li et al. (2016): we keep the maximum likelihood training intact, whereas in the inference phase, we use a pre-trained language model on InScript to estimate the anti-LM term and add it to the scoring within beam search. The coefficient λ is set at 0.1.

Overall, the objective function is

$$L(d) = -\sum_{tok_t:regular} \log p_t(token_t) - \sum_{tok_t:irregular} \log q_t(token_t) \tag{3.5}$$

Figure 3.10 gives an illustration of the model architecture.

3.3.4 Inference Behaviors

Agenda generation

In the inference phase, agendas are automatically generated by a bigram language model estimated on the event sequences taken from the corpus. We include irregular events in the event-LM so that the generated agendas include irregular events. Irregular events are less specific of the contents; instead of specifying the content of the next text segment as a regular event does, an irregular event simply marks the locations where additional details tent to be added.

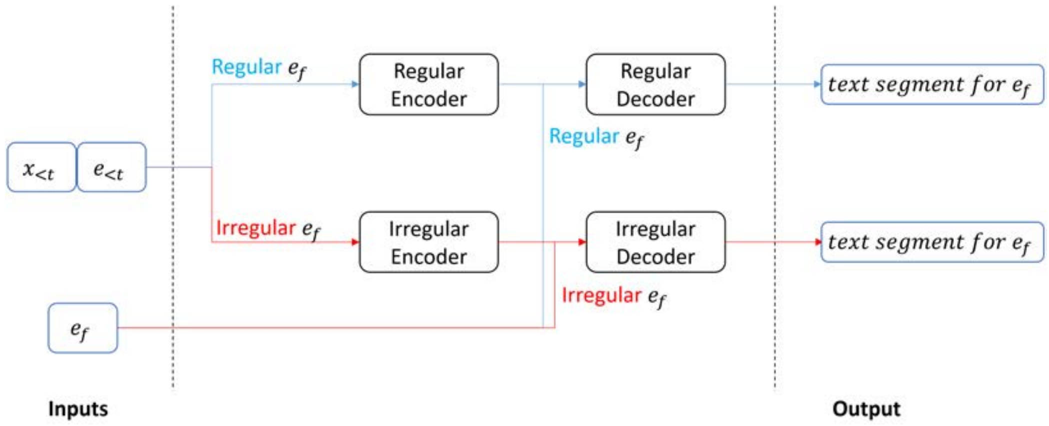


Figure 3.10: An illustration of our model that generates detail-rich stories. The figure shows the generation of the text segment corresponding to the forthcoming event e_f. The respective decoder is selected based on the regularity of e_f.

Segment generation

For better quality, text segments are generated with beam search. The beam size is set to 5 for *outliner*. For *detailer*, the situation is somewhat less trivial. Recall that we have an anti-LM term that is only taken into consideration in the inference phase. The anti-LM term contributes to the final scoring of the candidates in the beam search, in addition to the likelihood term. Two implications follow. First, the an-LM term must be evaluated during the beam search; therefore, the beam size cannot be too large for the sake of time complexity. Second, it could well be that some good candidate items have a moderate likelihood, yet advantageous, anti-LM term, which means, the number of candidates whose anti-LM term we evaluate should be large enough to grant a reasonable coverage. In practice, we set its beam size to 100 after some empirical observations.

3.3.5 Implementation and Optimization

5% of the InScript stories are randomly selected as the validation set. As we use human evaluation instead of automatic metrics such as the perplexity of the test set, no test set is necessary. The model is implemented with AllenNLP 0.9.0 (Gardner et al. (2017)). The hyperparameters were chosen by random hyperparameter search (Bergstra and Bengio (2012)). The model is optimized with Adam (Kingma and Ba, 2014) at a learning rate of 7.5×10^{-4}. The dimensions of the encoder and decoder are both 256. The word embedding size is fixed at 300 as we initialize it with the

840B version of the pre-trained Glove. A dropout (Srivastava et al. (2014)) at rate 0.69 is applied to all dense connections. Gradient norms are clipped at 2.0. We use early stopping with a patience of 35 to further regularize the training. Each training session takes on average 4 hours on a Tesla v100, whereas the generation speed is on average 8 tokens/second.

3.3.6 Evaluation

Baselines

We evaluate our model against three baselines: **(1)** the GRU-based model from Zhai et al. (2019b), as it operates on the same corpus, and also takes an ordered agenda as input. **(2)** an ablated version of our model, single decoder, which uses only one decoder, the *outliner*, to neutralize the design to treat regular and irregular contents differently. **(3)** the human-authored stories from InScript, to upper bound the performance. For some intuition, Table 3.5 provides a few sample texts.

Human Evaluation

We evaluate the output text using crowd-sourcing. Our evaluation has two purposes.

(1) As our main objective, we evaluate the amount of detail that a story includes, that is, how informative a story is about a specific experience. This is captured with an *informativeness* score.

(2) We want to ensure that the improvement in informativeness does not compromise the global coherence of the stories. Therefore, we assess whether a story is globally coherent as a narrative of its scenario, for example, going grocery shopping. This means that a story should incorporate common sense knowledge of going grocery shopping, including the necessary steps and their temporal order. This part involves five questions: *syntax* evaluates the basic syntax; *global coherence* evaluates common sense knowledge about the activity included in the story; *coverage* evaluates whether the story realizes each event in the agenda (for a human-authored story, we take its event annotations as its agenda); *relevance* evaluates whether the story stays on-topic; *local coherence* evaluates the flow of successive sentences, in terms of both content and fluency. It also evaluates whether the transition between two different decoders is smooth.

Our intention is to generate stories that contain rich detail and establish global coherence in the meanwhile. Reflecting on our evaluation, ideally, our model should achieve a decent coherence score, for example, somewhere on par with Zhai et al.

Zhai et al. (2019b)

yesterday i went grocery shopping . i made a list of my list and drove to the grocery store . when i entered the store , i grabbed a shopping cart and pushed the cart down to the meat aisle . i got all my items , and crossed items on my list . i went to the checkout register and paid for my groceries . i put my groceries in my cart and left .

Agenda: *evoking→make list→go to store→enter→take cart→move along sections→take grocery→check list→go to checkout→pay→pack grocery→leave→story ends*

Single Decoder

yesterday , i went grocery shopping . *i was very excited so i went grocery shopping .* i went to the store *and parked my car .* i grabbed a shopping cart so i went to the produce section . i went to the store so i picked up my list . i got off my list *when i was done* , i went to the cashier and paid for my debit card . i put my bags into my cart and paid for my groceries . then i paid , *i thanked the cashier* and headed to the car . i put the cart in the cart and drove home .

Agenda: *evoking →irregular go to grocery store→irregular→take a shopping cart→move along the sections→get groceries→check bought items off the shopping list→irregular→go to checkout→pay→irregular→pack groceries→get receipt→bring to vehicle→return the shopping cart→leave*

Full

i went grocery shopping today . i made a list of things that i needed before i left my house . i looked at the grocery store immediately , *i drove to the grocery store that is always empty .* i parked the car , walked inside and grabbed a shopping cart . i proceeded down the produce aisle , *which happened to be on sale .* i went to the back and chose my favorite peanut butter . i went to the cashier , which was a short line . i had to wait in line since there were some people ahead of me . i put my items on the conveyor belt and then she scanned my items . i said hi to the cashier , i paid my bill , and another man placed all my items in my shopping bags . i made sure to put the cart away , too . *once i had everything i needed ,* i then picked up my bags and left the store . *on the way out i decided to rent a redbox movie for the night since the kiosk was right by .*

Agenda: *evoking→make a shopping list →go to grocery store→irregular →take a shopping cart →move along the sections→irregular →get groceries→ go checkout→wait→cashier scan/weigh items→put stuff on the conveyor→irregular→pay→get receipt →wait →bring to vehicle→leave→wait*

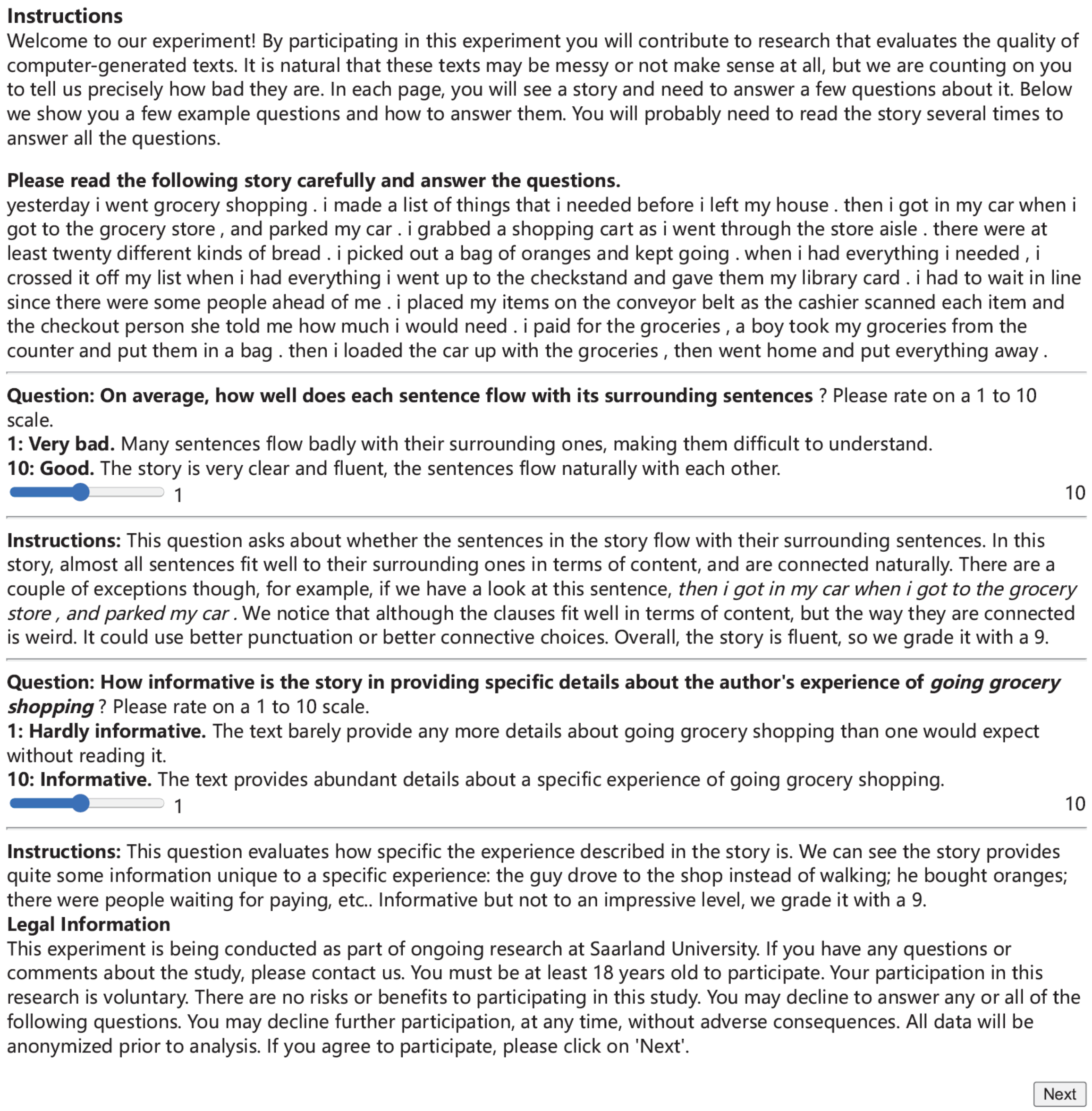
Human Author

yesterday i went grocery shopping . i took my grocery list with me , along with some reusable shopping bags . my grocery list has all the items i want to buy on it . i selected a shopping cart from in front of the store , and went inside . *i put my reusable bags in the cart .* i looked at my list and started in the produce section . i put different vegetables and fruits into my cart . next i wheeled my cart to the cereal aisle and took a box of cereal . i went through the store aisle by aisle and selected my groceries . *each aisle is organized by types of food and non-food items . one aisle has dried pasta , canned tomatoes , rice , and sauce* . i selected a few boxes of pasta and some rice . another aisle carries plastic wrap , trash bags , and aluminum foil . as i went through the store , i kept looking at my list to see what i needed next . when i added each item to my cart , i crossed it off my list . my last stop was the dairy aisle where i got milk and eggs . *when i had all the groceries i wanted ,* i went to the cash register ans stood in line . *when it was my turn ,* i put each item on the conveyor belt and the cashier scanned each one . a bagger put all of the groceries into my reusable bags . i paid , and then the cashier gave me a receipt . i loaded the bags of groceries into the trunk of my car and drove home .
Agenda: *evoking → take bags →get groceries →take a shopping cart →enter →irregular →check shopping list →get groceries→move along the sections →get groceries→move along the sections →get groceries→check items off the list →irregular→get groceries→irregular→wait →irregular→put stuff on the conveyor→cashier scan/weigh items →pack→pay→get receipt →bring to vehicle→leave*

Table 3.5: Grocery shopping stories generated by different models, together with respective agendas. The text corresponding to irregular events is italicized. We could see that the text produced by our model provides much richer details than the automatic baselines.

(2019b); in addition, a significant improvement in informativeness would validate our approach.

The evaluation experiment is implemented with LingoTurk (Pusse et al., 2016) and carried out on Prolific (`https://www.prolific.co/`). The questions are presented as slide-bars. We evaluated 4 stories per scenario per system and hired 10 native English speakers to score each story. Participants are paid on average 7 GBP per hour. See Figures 3.11, 3.12 for an illustration of the evaluation experiment.



Instructions
Welcome to our experiment! By participating in this experiment you will contribute to research that evaluates the quality of computer-generated texts. It is natural that these texts may be messy or not make sense at all, but we are counting on you to tell us precisely how bad they are. In each page, you will see a story and need to answer a few questions about it. Below we show you a few example questions and how to answer them. You will probably need to read the story several times to answer all the questions.

Please read the following story carefully and answer the questions.
yesterday i went grocery shopping . i made a list of things that i needed before i left my house . then i got in my car when i got to the grocery store , and parked my car . i grabbed a shopping cart as i went through the store aisle . there were at least twenty different kinds of bread . i picked out a bag of oranges and kept going . when i had everything i needed , i crossed it off my list when i had everything i went up to the checkstand and gave them my library card . i had to wait in line since there were some people ahead of me . i placed my items on the conveyor belt as the cashier scanned each item and the checkout person she told me how much i would need . i paid for the groceries , a boy took my groceries from the counter and put them in a bag . then i loaded the car up with the groceries , then went home and put everything away .

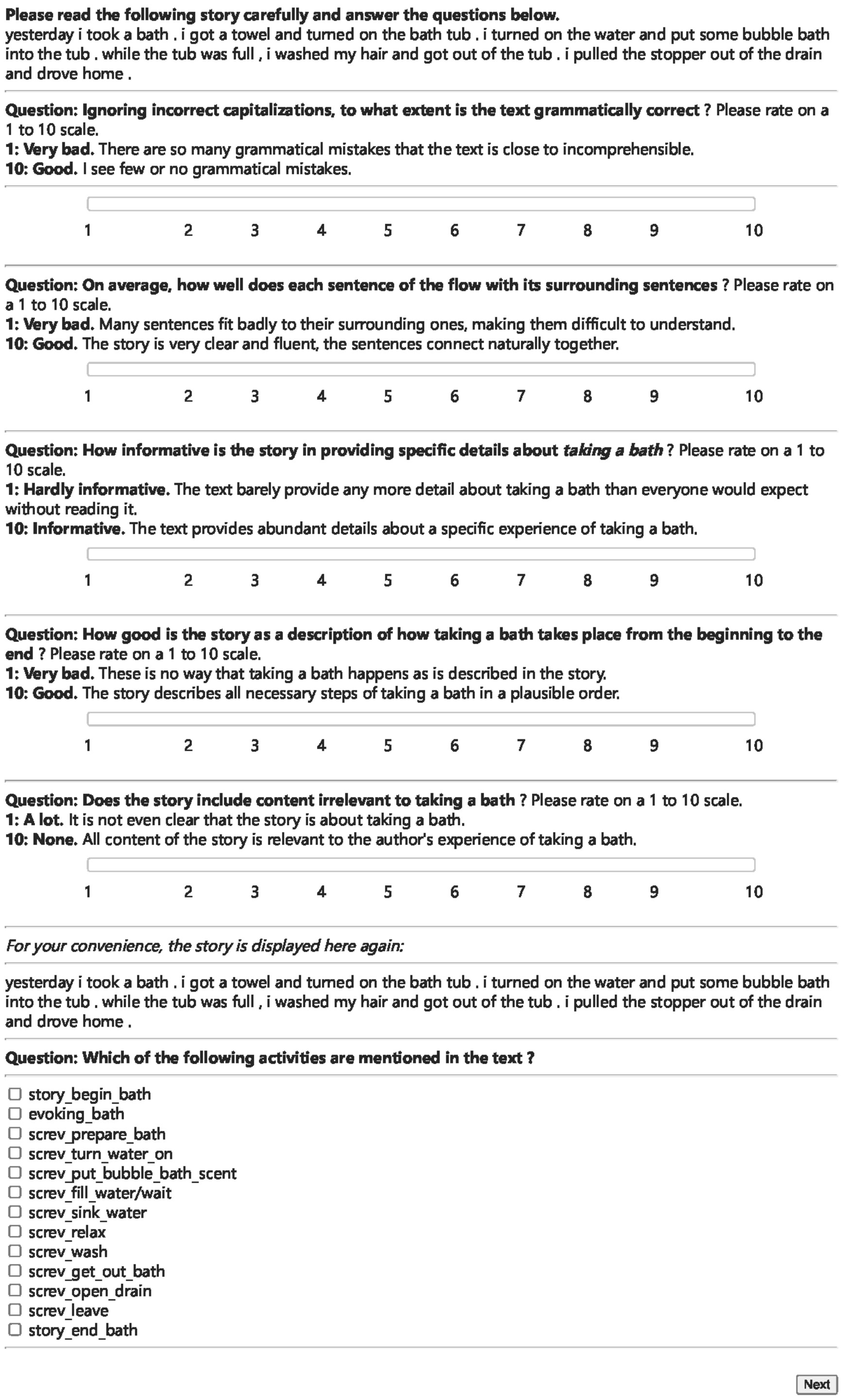
Question: On average, how well does each sentence flow with its surrounding sentences ? Please rate on a 1 to 10 scale.
1: Very bad. Many sentences flow badly with their surrounding ones, making them difficult to understand.
10: Good. The story is very clear and fluent, the sentences flow naturally with each other.

Instructions: This question asks about whether the sentences in the story flow with their surrounding sentences. In this story, almost all sentences fit well to their surrounding ones in terms of content, and are connected naturally. There are a couple of exceptions though, for example, if we have a look at this sentence, *then i got in my car when i got to the grocery store , and parked my car* . We notice that although the clauses fit well in terms of content, but the way they are connected is weird. It could use better punctuation or better connective choices. Overall, the story is fluent, so we grade it with a 9.

Question: How informative is the story in providing specific details about the author's experience of *going grocery shopping* ? Please rate on a 1 to 10 scale.
1: Hardly informative. The text barely provide any more details about going grocery shopping than one would expect without reading it.
10: Informative. The text provides abundant details about a specific experience of going grocery shopping.

Instructions: This question evaluates how specific the experience described in the story is. We can see the story provides quite some information unique to a specific experience: the guy drove to the shop instead of walking; he bought oranges; there were people waiting for paying, etc.. Informative but not to an impressive level, we grade it with a 9.
Legal Information
This experiment is being conducted as part of ongoing research at Saarland University. If you have any questions or comments about the study, please contact us. You must be at least 18 years old to participate. Your participation in this research is voluntary. There are no risks or benefits to participating in this study. You may decline to answer any or all of the following questions. You may decline further participation, at any time, without adverse consequences. All data will be anonymized prior to analysis. If you agree to participate, please click on 'Next'.

Figure 3.11: The instructions for the evaluation experiment. Participants are asked to confirm that they have read the instructions before proceeding.



Please read the following story carefully and answer the questions below.
yesterday i took a bath . i got a towel and turned on the bath tub . i turned on the water and put some bubble bath into the tub . while the tub was full , i washed my hair and got out of the tub . i pulled the stopper out of the drain and drove home .

Question: Ignoring incorrect capitalizations, to what extent is the text grammatically correct ? Please rate on a 1 to 10 scale.
1: Very bad. There are so many grammatical mistakes that the text is close to incomprehensible.
10: Good. I see few or no grammatical mistakes.

1 2 3 4 5 6 7 8 9 10

Question: On average, how well does each sentence of the flow with its surrounding sentences ? Please rate on a 1 to 10 scale.
1: Very bad. Many sentences fit badly to their surrounding ones, making them difficult to understand.
10: Good. The story is very clear and fluent, the sentences connect naturally together.

1 2 3 4 5 6 7 8 9 10

Question: How informative is the story in providing specific details about *taking a bath* ? Please rate on a 1 to 10 scale.
1: Hardly informative. The text barely provide any more detail about taking a bath than everyone would expect without reading it.
10: Informative. The text provides abundant details about a specific experience of taking a bath.

1 2 3 4 5 6 7 8 9 10

Question: How good is the story as a description of how taking a bath takes place from the beginning to the end ? Please rate on a 1 to 10 scale.
1: Very bad. These is no way that taking a bath happens as is described in the story.
10: Good. The story describes all necessary steps of taking a bath in a plausible order.

1 2 3 4 5 6 7 8 9 10

Question: Does the story include content irrelevant to taking a bath ? Please rate on a 1 to 10 scale.
1: A lot. It is not even clear that the story is about taking a bath.
10: None. All content of the story is relevant to the author's experience of taking a bath.

1 2 3 4 5 6 7 8 9 10

For your convenience, the story is displayed here again:

yesterday i took a bath . i got a towel and turned on the bath tub . i turned on the water and put some bubble bath into the tub . while the tub was full , i washed my hair and got out of the tub . i pulled the stopper out of the drain and drove home .

Question: Which of the following activities are mentioned in the text ?

☐ story_begin_bath
☐ evoking_bath
☐ screv_prepare_bath
☐ screv_turn_water_on
☐ screv_put_bubble_bath_scent
☐ screv_fill_water/wait
☐ screv_sink_water
☐ screv_relax
☐ screv_wash
☐ screv_get_out_bath
☐ screv_open_drain
☐ screv_leave
☐ story_end_bath

Figure 3.12: A sample page of the evaluation experiment.

Results

	Coverage	Syntax	Global Coh.	Local Coh.	Relevance	Informativeness
human author	76%^{fsz}	.62^{fsz}	.66^{fsz}	.68^{fsz}	.60^{fsz}	**.66**^{fsz}
full	63%^{s}	.43^{s}	.49	.42	.59	**.51**^{z}
single decoder	57%	.38	.42	.38	.52	.47^{z}
Zhai et al. (2019b)	66%^{fs}	.45^{s}	.40	.44^{s}	.54	.38

^{f},^{s} ,^{z}: improvement over the respective system is statistically significant according to paired T-test at $\alpha = 0.05$.

Table 3.6: Results from human evaluation. All metrics are ranged $[0, 1]$.

The results are given in Table 3.6. We see that on almost all metrics, human author outperformed all other systems by a large margin, fulfilling its role as an upper bound of performance. For our full model, we see that both conditions are met to validate its functionality: (1) it outperformed Zhai et al. (2019b) in *informativeness* significantly and by a large margin, indicating that the generated stories include much richer details about specific experiences; (2) it performed on par with Zhai et al. (2019b) on the first five metrics, which means that the stories it generates follow their agendas and describe script instances in a plausible manner, establishing decent global coherence. We also see that full outperformed single decoder in all metrics, supporting our point that story writing decomposes into outlining and detailing, and that the model benefits from addressing them differently.

3.3.7 Summary

In this section, we present a story generation system that includes details in the stories it generates. These stories are script stories that follow the story lines specified in their agendas. The model determines the content of the details it adds by tracking the progress of the instantiation of the narration. The effectiveness of our approach proves that script knowledge benefits the establishment of text coherence, not only when guiding the generation to follow given story lines but also in the selection of contents that focuses on specific steps of the given story line.

Chapter 4

Supervised Neural Script Parsing

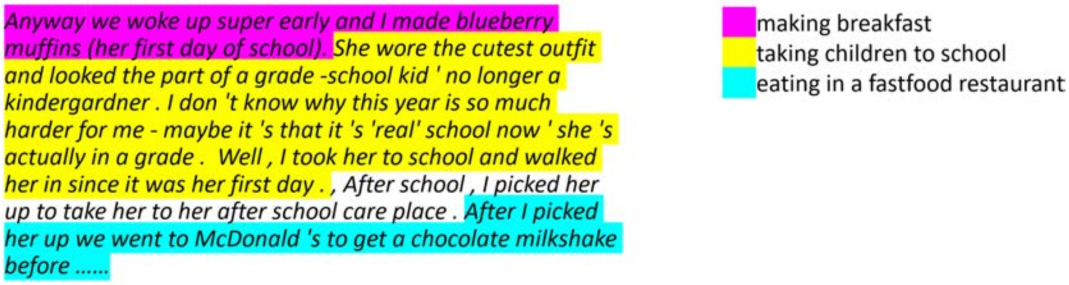
4.1 Introduction

4.1.1 Task

Figure 4.1: The task. Given descriptions of fixing a flat tire from InScript, script parsing identifies events and participants from surface text.

a. *The bus drivers turned out to be on strike today. I had to take a taxi to work.*

b.



Anyway we woke up super early and I made blueberry muffins (her first day of school). She wore the cutest outfit and looked the part of a grade -school kid ' no longer a kindergardner . I don 't know why this year is so much harder for me - maybe it 's that it 's 'real' school now ' she 's actually in a grade . Well , I took her to school and walked her in since it was her first day . , After school , I picked her up to take her to her after school care place . After I picked her up we went to McDonald 's to get a chocolate milkshake before ......

🟪 making breakfast
🟨 taking children to school
🟦 eating in a fastfood restaurant

Figure 4.2: Examples. a. scenario detection. Scenarios taking a taxi, taking a bus and going to work are all evoked in the text. b. A personal story segemented according to the scenarios each sentence is about.

Aligning surface text and its semantics is a central topic in NLP. **Script parsing** identifies the tokens in the surface text that evoke an event or a participant in the scenario of interest. Figure 4.1 shows the task. Given a story and our scenario of interest fixing a flat tire, what script parsing does is finding tokens that evoke script events or participants, and determining which specific events or participants they evoke.

A token that potentially evokes an event is termed an **event candidate**; likewise, a token that potentially evokes a participant is termed a **participant candidate**. Representing an action, an event candidate is usually a verb; a participant candidate, which represents an entity, is always the head of a noun phrase. These candidates could be identified according to shallow linguistic features such as syntactic trees. The non-trivial component of the task is labeling these candidates with the event or participant they evoke, which is usually formulated as a sequence labeling task. In the next section, we introduce a linear-CRF-based approach (Ostermann et al., 2017); in section 4.3, we show how the task could be approached with neural networks which reaches much higher accuracy.

Script parsing is a complex process. We have a long way to go from a narrative to tokens that evoke events or participants of a specific scenario. In general, it consists of multiple subtasks. Consider the example in Figure 4.1. The pipeline includes the following steps.

1. **Scenario detection.** We are to identify tokens that suggest key elements of scenario fixing a flat tire. First of all, we need to determine whether our scenario of interest is evoked at all in this piece of text. In our example, there are no direct mentions of fixing a flat tire. However, we could infer from the actions performed by the protagonist that fixing a flat tire is evoked. Scenario detection could also be easier if the scenario itself is directly mentioned. Consider Example a. in Figure 4.2. Clearly, scenario taking children to school is directly mentioned here.

2. **Text Segmentation.** For each scenario evoked in our text, text segmentation decides which parts of the text are devoted to each scenario. These segments contain the event and participant mentions of its corresponding scenario; within these segments, we will be looking for the event and participant evoking tokens. Figure 4.2b. shows how a personal story is segmented according to the scenarios each sentence is devoted to. It could also be that the text assigned to each scenario overlaps.

3. **Identification of events and participants.** Given a scenario and a segment of text on it, the last step is to find out the tokens that evoke an event or a participant of that scenario, and determine the specific event or participant type evoked by them. The sets of events and participants are specific to each scenario. Conceptually, they should match what is stored in human memory.

> In Figure 4.1, we see that all tokens that evoke an event or a participant are labeled.

Script parsing bridges script knowledge and surface texts. It grants NLP applications convenient access to script knowledge. We could learn which scenarios are mentioned, what participant or event of that scenario is of interest, etc. In the scope of this thesis, we focus on the last step, that is, identifying event or participant evoking tokens from a segment of text that is known to be about a scenario of interest. The current chapter is on supervised script parsing. In supervised settings, we work with the same set of scenarios during the training and the inference phase. That means, the task is to parse a piece of text with regard to a *known* scenario, of which we know the sets of events and participants, and we have training data to model the patterns in which they are evoked in surface text.

4.1.2 Technical Challenges

Script parsing is a tough task which is challenging even for humans: the inter-annotator agreement is quite modest, at 0.64 and 0.77 Fleiss' κ for event and participant parsing, respectively (Modi et al., 2016). Here, we discuss a few underlying factors that make the task complicated.

The variety in surface language

Script events are usually evoked by verbs in the surface text. However, the correspondence between events and verbs is not trivial. A script event is an abstraction of all possible activities that is considered equivalent with regard to a specific scenario. Although some events tend to be evoked by a typical verbs, it is not uncommon for an event to see a vast difference among the verbs that possibly evoke it. For example, in the taking a train scenario, its spend time in the train event refers to the activities with which the passengers spend the time during the ride. These activities see various surface realizations, such as reading a book, taking a nap, having lunch, looking out of the window, etc. These activities are considered equivalent in this scenario, and thus they correspond to the same event.

The same argument holds for script participants. For example, ingredients in baking a cake could be evoked by various different words such as *flower*, *butter*, *the ingredients*, and *all I need*.

Granularity

Granularity matters in various occasions. First, the scenarios. Many scenarios could comprise several *smaller* scenarios. As an example, we might consider going to work as a scenario of interest. However, many other scenarios could be contained in the course of going to work, such as taking a bus, dressing up for work, etc.

Apart from scenarios, the designation of events and participants is also subject to the granularity issue. For example, in the cooking dinner scenario, we could have utensils as a participant; it is also reasonable to have plates, knives, etc. Or maybe only kitchen tools. We may have prepare ingredients, or more fine-grained steps such as chopping a garlic or marinating a fish. Each different decision would mean a different way of classifying the evoking words. Determining the set of events and participants is also a most complex component in the annotation of InScript (see Modi et al., 2016).

Context

Script parsing is more than a local task. Lower-level semantics do matter, as, for example, we look for participant evoking words only among the heads of the noun phrases. Also, when an NP is a dependent of an adjacent verb, the predicted participant type must be one that participates in the predicted event. On a higher level, the sequence of event and participant labels produced by a script parser must suggest a plausible agenda, which makes the whole narrative relevant. Labeling of one candidate must take into account all other labels. One important factor is the temporal order among script events; for most scenarios, events occur in prototypical orders. It could be expected that a piece of narrative text roughly follows temporal order, at least after adjustments were made based on discourse relations. However, this is largely dependent on the genre of the text. A user manual would follow the order in which operations could be made; a personal blog might begin with the most interesting part of the experience, whereas most others were omitted.

What makes the situation even more complicated is the fact that events and participants are frequently omitted from a narrative. One might expect that the most important elements of the scenario are mentioned, like cake in baking a cake. Although the instantiation of a scenario already implies multiple events having occurred in some possible world, we usually find many events hidden from the text. The same applies to the participants. In our example 4.2 a., presumably, the passenger *paid* the taxi driver. Yet, neither *paying* nor *the taxi driver* is mentioned in the text. Humans tend to omit some well-anticipated events even when instructed otherwise.

4.2 Script Parsing with Linear CRF

In this section we introduce the state of script parsing before this thesis. Identification of script events was previously approached by Ostermann et al. (2017) with a two-stage method. In the first stage, they identify verbs that evoke a script event; in the second stage, they label these verbs with their respective event types.

4.2.1 The Identification of Script Relevant Verbs

Not all verbs evoke a script event. When telling a story on a script scenario, the narrator tends to mention activities that are not related to the scenario per se but are present in the specific instantiation he describes. These verbs were termed *script-irrelevant* verbs; they do not evoke any script event and need to be excluded from event parsing. Ostermann et al. (2017) identifies script relevant verbs with a decision tree. It is trained on InScript, while also exploiting information from DeScript. The features include the following.

Syntactic features. The syntactic features include auxiliaries, number of direct and indirect objects, and a predefined list of non-action words. With these features, most light verbs could be excluded. Note that these features are independent of the scenario.

Script features. The script features include a verb-scenario tf-idf evaluated on DeScript, which directly models the relevance between verbs and scenarios. The authors also include a binary feather that indicates whether the verb appears in any DeScript ESD that belongs to that scenario.

Frame features. The frame features are determined by the frame assigned to the verb, according to semantic role labeling results[1].

The result of the first stage is given in Table 4.1. Here, the baseline Lemma decides a verb is relevant if, and only if, it appears in the DeScript ESDs of the scenario,

---

[1] A *frame* is a pattern that specific semantic elements are composed to become a single semantic unit. A frame vaguely corresponds to a verbal sense that admits specific semantic roles. See, e.g., Ruppenhofer et al. (2016)

Model	Precision	Recall	F1
Lemma	36.5	94.9	52.6
Full	62.8	81.7	70.9
w/o Script Features	51.3	87.7	64.5

Table 4.1: Results of script-related verb identification from Ostermann et al. (2017)

which exhibits high recall and low precision. w/o Script Features does not use the script features, which sees a significant performance drop from Full. This is anticipated, as some knowledge that script parsing depends on is scenario-specific. full achieves 70.9 points F1 score, which is sufficient for the next step.

4.2.2 Classifying Event Types

With script relevant verbs identified, the second step is to classify them into event types. This task is contextualized. Ostermann et al. (2017) formulate it as a sequence labeling task, which they approach with a linear CRF model. It uses two categories of features.

Sequential features. These features encode co-occurrence information of event types in DeScript ESDs.

Meaning representation features. These features encode information from the embeddings of the key parts of the predicate-argument structure with the verb of interest at its head. These features are generated by pooling the word vectors of the verb, its direct nominal dependents, its lemma, and its direct or indirect object.

The model is trained on DeScript and evaluated on InScript. The result is shown in Table 4.2.

Finally, both stages are assembled as a complete pipeline. The pipeline is evaluated as a whole in Table 4.3; the complete model achieves a score of 47.8 points F1. The results are decent, but further improvement is needed before it could offer practical usefulness.

Ostermann et al. (2017) is the first attempt at script parsing. They insightfully formulated script parsing as a sequence labeling task over the entire narrative

Model	Precision	Recall	F1
Lemma	51.6	44.2	47.5
Word2Vec	53.8	48.0	50.7
CRF	**62.3**	**48.5**	**54.3**
CRF w/o Script Features	60.8	47.5	53.1

Table 4.2: Results of event type classification from Ostermann et al. (2017). Here, lemma is based on word overlap with the event types; Word2Vec is based on cosine similarities with the event types.

Model	Precision	Recall	F1
Full+Lemma	38.8	47.5	42.6
Full+Word2Vec	39.3	**51.1**	44.2
Full+CRF	**45.8**	50.5	**47.8**

Table 4.3: Results of event type classification from Ostermann et al. (2017)

and delivered promising results. In the next section, we show how script parsing could be performed accurately with refined sequence modeling and neural network technologies.

4.3 Script Parsing with Hierarchical Sequence Modeling.

In this section, we propose a new script parser that delivers better performance. Methodologically, our new parser introduces the following ingredients.

Hierarchical sequence modeling. Script parsing is a task for which we need to consider semantics at different levels. To that end, we use two different sequence models, one for the sequence of tokens, the other for the sequence of script events and participants.

Neural network methods. Ostermann et al. (2017) is based on symbolic methods, with the exception that they incorporated some information from pretrained word vectors. Here, we approach the whole process with and end-to-end neural model.

Jointly parsing script events and participants. Script participants were not addressed by Ostermann et al. (2017). Here, we simultaneously perform event parsing and participant parsing. Our higher-level sequence model also addresses both script events and participants. In addition to enabling us to parse script participants, the extended sequence is also more informative to event parsing.

This section develops previous research from two perspectives. First, the model performs more accurate event parsing, reaching an average F1 score of 86 points, improving the previous state-of-the-art by at least 17 points; second, the model accurately performs participant parsing for the first time, delivering 90 points average F1.

4.3.1 Data

Our work uses both InScript and DeScript (see Chapter 2 for more details on the corpora). InScript provides most of our training data; it is also where we evaluate our models. DeScript is used as auxiliary training data, which alleviates InScript's data sparsity issue. Compared to InScript, DeScript is very concise, where regular events take a much larger proportion: 12,902 (33.5%) event candidates in InScript evoke a regular event, while the numbers are 4,185 (89.1%) for DeScript.

Given a story, we label the tokens with the event or participant types they evoke, if any. The annotations of InScript provide the ground-truth labels.

InScript scenarios vary in their structures. This is reflected in the class sizes of regular events and regular participants. On average, each scenario has 19.2 regular event classes and 18.9 regular participant classes. taking a flight has the largest class sizes for events and participants (29 and 26, respectively), while planting a tree has the smallest (14 and 15, respectively). These event clusters also have quite different sizes, which imposed some technical challenge to modeling; see Section 4.3.3.

4.3.2 Model

Overall

We train a scenario-agnostic model that analyzes all InScript scenarios. Conceptually, our model implicitly contains a *scenario detection* component, which determines the scenario the input text is about. Our model consists of two sequence models: **(1)** a *word sequence model* and **(2)** an *event sequence model*.

The word sequence model addresses the sequence of tokens in the input. Although we find multiple aspects of script parsing beyond the sentence level, many features below the sentence level are indeed relevant when it comes to identifying an event or a participant. The lemma of a verb, its reflections, its dependents, etc., could encode vital clues about what event type it could evoke. Similarly, the lemma of a noun, the verb or preposition that governs it, etc., are also informative about what type of participant it potentially evokes.

The event sequence model processes the sequence of all event candidates and participant candidates, which consists of all the verbs and heads of the noun phrases. This *event sequence*, collecting the activities and their participants, offers a higher-level view of the story line. By modeling this sequence we capture patterns on the event level, such as typical preceding and successive event, the interaction among events and participants, as well as global patterns like what events are always present in the sequence.

Figure 4.3 illustrates the model architecture. The input consists of a story x from the corpus d and an ordered set of indices I that specifies the positions of the candidates. It assigns an event / participant label to each of these candidates as output. The sets of labels are predefined in InScript and specific to each scenario. The set of candidates consists of all NPs and verbs in the text. We use the InScript tokens as annotated in it; yet, they could also be extracted with a syntactic parser.

Technical Design

The **word sequence model** encodes the entire story as a list of vectors. This could be approached by contextualized word embeddings. What we use is a pre-trained XLNet model, *xlnet-base-cased* (Yang et al., 2019):

$$\tilde{x} = \mathrm{XLNet}(x) \tag{4.1}$$

Next, to accommodate sequence modeling at the discourse level, we only keep the representations in $\tilde{x}$ that correspond to the candidates, namely, the NP heads and the verbs. Their positions in $\tilde{x}$ are specified by the ordered sequence of indices I. We directly select the vector representation of these tokens and ignore the rest to form a subsequence of $\tilde{x}$, the **event sequence**. This operation is termed **index_select** in the Pytorch library:

$$c = \tilde{x}.index\_select(I) \tag{4.2}$$

Now that we have the representation of the sequence of events and participants, which is a sketch of the story at a higher level, the flow of tensors is ready to be processed by the event sequence model ψ, which is a bidirectional LSTM:

$$\tilde{c} = \psi_{\theta_\psi}(c) \tag{4.3}$$

Finally, $\tilde{c}$ is fed to a linear classifier γ to produce the output distribution for each candidate:

$$p(y|x, I; \theta_\psi, \theta_\gamma) = softmax(\gamma_{\theta_\gamma}(\psi_{\theta_\psi}(c))) \tag{4.4}$$

The model does not distinguish between event types and participant types. This is intended: including both event items and participant items in the event sequence results in better informed event sequences, which is important to the event sequence model. The event candidates are easy to distinguish from the participant candidates, so no significant learning load is introduced.

We train the model by optimizing data likelihood:

$$\theta^* = \operatorname*{argmax}_{\theta_\psi, \theta_\gamma} \sum_{x,I} log(p(y|x, I; \theta_\psi, \theta_\gamma)) \tag{4.5}$$

4.3.3 Addressing Data Sparsity

The size of InScript is limited by its high cost. Its size is 234k tokens, which could barely accommodate the optimization of our model, but also leaves some consequences. Most importantly, the class distribution is skewed and some classes

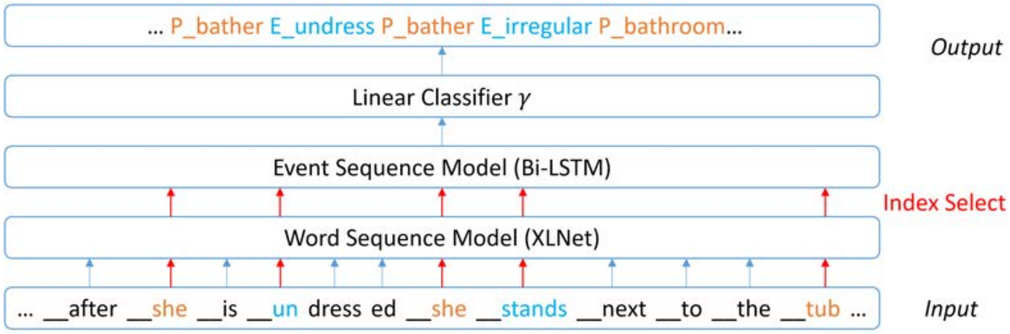


Figure 4.3: The model architecture. Note that index selection is performed before the Bi-LSTM layer.

are quite small: the largest regular event class has 397 instances (get groceries in Grocery shopping), while there are 26 classes with less than 10 instances (e.g., get receipt in Grocery shopping, scalp massage in taking a bath). These classes are difficult to learn reliably. In fact, since the proportions of our validation set and our test set are both 10%, it is not even guaranteed that these classes have an instance in both the validation set and the test set. Therefore, validation is not well defined on these classes.

Our efforts to address data sparsity are two-fold. We use DeScript as auxillary training data; in the meanwhile, we also automatically generate additional training data via paraphrasing.

Domain Adaptation from DeScript

DeScript consists of parallel event sequence descriptions for each of 40 scenarios. It does not include participant annotations, however, according to Ostermann et al. (2017), the knowledge encoded in DeScript is beneficial to event parsing. More importantly, all 10 InScript scenarios are included in DeScript. These facts motivate us to transfer the knowledge encoded in DeScript to improve script parsing on InScript.

We directly use DeScript as additional training data. To address the domain difference, we experiment with two domain adaptation methods. Firstly, **(1)** *Data concatenation*, which concatenates DeScript with the original InScript train set. It is a most straightforward approach, however, it might not be the best idea, as the domain gap is also noticeable. Descript consists of short telegram-style event descriptions, most of which are mere verbal phrases, such as *preheat your oven*, *put the dough in the over* and *enjoy your cake*, whereas InScript consists of complete sentences. Moreover, since they are from personal stories, these sentences could include information that is not as closely related to the scenario as any DeScript phrase. Therefore, we also experiment with explicit domain modeling which features **(2)** *Corpus embeddings.* We follow Stymne et al. (2018) to train a vector representation (the **corpus embedding**) for each corpus to capture corpus-specific patterns. We concatenate a copy of the corpus representation with the representation of each candidate. Equivalently we substitute the classification features ($\tilde{c}$) for the linear classifier γ with

$$\hat{c} = \psi_{\theta_\psi}(c; \eta_{\theta_\eta}(d)) \tag{4.6}$$

Here $\eta(\cdot)$ denotes the corpus embeddings.

Data Augmentation via Paraphrasing

We generate additional training data by paraphrasing InScript texts. In particular, we paraphrase with *back-translation* (see, e.g., Bojar and Tamchyna, 2011; Sennrich et al., 2016; Xie et al., 2020). In particular, we use pre-trained translation models to translate the training data to a different language and translate the translation back to the original language. The resulting text is similar to the original and could be considered as its paraphrase, if the translation model delivers decent performance. If the original annotation could be restored, the backtranslated text could be used as additional training data. Translation processes generally introduce legitimate linguistic variations to the original, which improves the generalizability of the model.

We translate the stories into French and back to English with Google Translate. The choice of French is because the back-translated text admits better quality than the other options we experimented with.

The annotations in InScript are at the word level, which means that restoring annotations for the paraphrased text is not trivial, as the translation runs story by story. We aligned the backtranslated data with original InScript stories in terms of events and participants to translate the annotations. The alignment relies on string and position matching, as well as synonyms and similar words to map the event and participant labels from the originals to the paraphrases. If the labeled source token or its lemma appears in the same position in the original as in the backtranslated story, we copy the label. We also consider similar words regardless of the position. We search for paraphrases using WordNet (Fellbaum, 1998) by considering the synonyms of the labeled source words. For each labeled source word, we also consider the 10 closest vectors found in the embedding space of word2vec-google-news-300 pre-trained vectors (Řehůřek and Sojka, 2010) to see if any of these neighboring

Original: when I_{rider} was $riding_{ride}$ my_{rider} $bike_{bike}$ this past summer ...
French: l'été dernier, je montais mon vélo ...
Paraphrase: when I_{rider} $rode_{ride}$ my_{rider} $bike_{bike}$ this summer ...

Figure 4.4: Example. Paraphrase with back-translation to and back from French. We transfer the original annotations to the paraphrased sentences with rule-based methods. The restored annotations are marked in the paraphrased sentence.

words appear in the backtranslated story. For the sake of quality, we do not map all labels from the original to the back-translation, but attempt to trade recall for precision. As a result, the paraphrase dataset contain 18% fewer event labels and 10% fewer participant labels. The new data was concatenated with the original InScript to yield a larger dataset which we consider homogeneous. Below is an example sentence with its translation to French and its back-translation to English, together with the restored annotations. In this example, the event verb takes a different tense and surface form in the back-translation, which would, among others, improve the generalization of the model to different reflections of verbs.

4.3.4 Implementation

For each scenario, 80%, 10%, 10% of the stories are randomly selected and designated as the training, validation and test sets, respectively. The model is implemented with AllenNLP 1.0 (Gardner et al., 2017). To regularize the model, dropout (Srivastava et al., 2014) with a universal rate is applied to all dense layers, in conjunction with weight decay. We use gradient norm clipping to stabilize the training. The optimization is performed with adam (Kingma and Ba, 2014) in conjunction with early-stopping which monitors validation loss, and the hyperparameter tuning is performed with random hyperparameter search (Bergstra and Bengio, 2012). Optimization takes on average 3.5 ours on a singe Tesla v100. We performed 20 trial for choosing the hyperparameters, and 5 parallel optimizations in order to perform the significance test. Table 4.4 gives our choice of hyperparameters.

4.3.5 Evaluation

Ablations and Baselines

We have two external baselines. First, we compare with the SotA model from Ostermann et al. (2017); secondly, we include fine-tuned XLNet as a baseline, as it is interesting to see what performance an off-the-shelf, pre-trained large model delivers. For this baseline, we fine-tune the pre-trained XLNet to perform script

learning rate	7.3×10^{-5}
batch size	32
weight decay	0.001
dropout	0.57
gradient clip	3.21
dimensionality	512

Table 4.4: The final choice of hyper-parameters.

parsing. The version of XLNet is *xlnet-base-cased*, same as what we used to encode the input tokens. We also include different versions of our model in the evaluation. **Hierarchical** is the full hierarchical model, as is described in formulas (1)-(4). For **no_index_select**, we ablate the index selection (thus $c = x$) to neutralize the event sequence model. Its event sequence model now takes every token of the story as input; therefore, the event sequence model still operates at the token level. The variants **concatenate** and **corpus_embedding** are the ones that exploit DeScript with the respective domain adaptation methods. Some models have access to the paraphrased dataset; these variants are marked with BT.

Results

Script parsing is evaluated with average F1 scores. The results are shown in Table 4.5. Most noticeably, a fine-tuned XLNet already outperforms **Ostermann** by more than 10 points of average F1 score. The gap is anticipated, which is yet another witness of how much performance improvement is delivered by pre-trained language models. If we look further down the table, we see that our model variants deliver further substantial improvements over XLNet. All our models outperform **Ostermann** and **fine-tuned XLNet** by a considerable margin, in both event macro and micro F1 scores. Furthermore, we perform participant parsing for the first time and with high accuracy.

On the addition of the event sequence model, **hierarchical** sees substantial improvements over **no_index_select**, which fails to perform sequence modeling at the discourse level. We also note that **hierarchical** improved both micro-F1s. Analysis shows that hierarchical models are generally better at addressing the most frequent yet problematic class, irregular candidates. These candidates do not participate in the core event chain, a decision that should be made after taking into account the structure of the candidate sequences. That is exactly the job of our event sequence model.

	Model	Train set	Events		Participants	
			Macro-F1	Micro-F1	Macro-F1	Micro-F1
1.	Ostermann	I	58.1	66.0	n/a	n/a
2.	fine-tuned XLNet	I	62.1^{1}	79.3^{1}	79.7^{345}	77.2
3.	no_index_select	I	63.3^{1}	78.3^{1}	74.3	87.1^{2}
4.	hierarchical	I	70.1^{123}	83.7^{123}	78.7^{3}	89.3^{23}
5.	concatenate	I, D	69.3^{123}	82.5^{123}	79.1^{3}	89.9^{23}
6.	corpus_embedding	I, D	74.9^{1-5}	82.9^{123}	78.6^{3}	89.4^{23}
7.	hierarchical^{BT}	I^{+BT}	$\mathbf{75.1}^{1-6,8}$	$\mathbf{85.7}^{1-6,8}$	80.3^{1-6}	$\mathbf{90.3}^{1-6,8}$
8.	corpus_embedding^{BT}	I^{+BT}, D	74.3^{1-5}	83.8^{1-6}	$\mathbf{80.9}^{1-6}$	89.5^{1-5}

$^{1-8}$: performance improvement over the respective model is significant at $\alpha = 0.05$ according to the independent T test.

Table 4.5: Results. The best of each column is displayed in boldface. Thanks to a larger training set, the optimizations of **7.** and **8.** is quite stable compared to the rest; thus, their performance differences compared to others are statistically significant, despite small effect sizes.

We also see that all micro F1 scores are noticeably higher than the respective macro F1 scores. This difference is due to the data including many small classes that are, in general, harder to learn. Indeed, we observe positive Pearson correlations between F1 and class sizes(around 0.5, all model variants included, $p < 10^{-10}$).

Participant parsing yields much higher scores than event parsing. The reason is straightforward: participant parsing is a relatively less complicated task compared to event parsing. There are several reasons. First, a large proportion of errors come from *irregular* candidates (see also Section 4.3.6). However, irregular participants (19.6%) are much fewer than irregular events (66.5%), making the former an easier task in general. Second, many participant candidates refer to the protagonist (31.0%), whereas the protagonist is almost exclusively evoked with a first-person pronoun. Thus, it is very easy to identify this class, which takes a large proportion of the data.

Now, we turn our attention to the domain adaptation methods. First, the comparison between concatenate and corpur_embedding suggests that corpus_embedding is the better way to exploit DeScript. This is anticipated, given the apparent domain gap, like the difference between the language styles of both corpora. hierarchical^{BT} sees a larger improvement over hierarchical as it has paraphrased InScript as additional training data, which is larger and a more similar domain than DeScript to InScript. These improvements over hierarchical are more noticeable in event macro-F1s, which means that these models are generally better at tagging smaller event classes, fulfilling our original goal of alleviating the problems caused by the uneven class sizes. Further addition of DeScript on top of hierarchical^{BT} (model 8.) does not deliver further significant improvement, but sees, overall, a modest performance drop. We suspect that, since the addition of backtranslated data, together with the original InScript, already creates a larger, relatively homogeneous training set, the domain difference between DeScript and InScript is beginning to outweigh the benefit of having DeScript.

4.3.6 Error Analysis

We perform error analysis by manually classifying the validation set errors made by our best-performing model, hierarchical^{BT}, case by case. A breakdown is presented in Table 4.6. Most error cases fall into the following three types. Some of these result from the inherent features of the task; others suggest directions to further improve our model.

Noisy Corpus Labels

Firstly, some annotations in the original corpus seem debatable. Although this might be inevitable for all annotated corpora, given the complexity of script parsing, these debatable annotations might be more frequent in InScript. In fact, we observe debatable annotations in a considerable proportion of the instances where our model errs. For example, in a story about borrowing a book from a library, *... I had to **get** a library card ...* is a clear match for the event obtain card, as predicted by our model; but in the corpus it was annotated as *irregular*, a mistake probably due to the light verb 'get' seemingly irrelevant to the scenario at first glance by the original annotator.

False Positives of irregular.

A large proportion of errors show a incorrectly predicted irregular. We identified two main sources of such errors. First, small class sizes. Given the large size of irregular classes, when the training signal is weaker, the model tends to give the generic prediction. Second, there are some instances that are particularly difficult because pragmatic inference is needed to make the right decision. As an example, *... **get** materials for the assignment ...* corresponds to the event class evoking_library, that is, it evokes the scenario of borrowing a book from a library without explicitly referring to any script event specific to that scenario. However, without taking into account the situational context of the scenario, it cannot be inferred that '***get*** *materials for the assignment*' actually mean '*borrow a book from a library*'.

Wrong Category.

A small number of events are tagged as participants and vice versa, for example, some homonyms of verbs and nouns (*board* or *love*).

Type	Events	Participants
Noisy corpus label	23%	26%
False irregular predictions	49%	37%
Wrong category	2%	4%
Others	26%	33%

Table 4.6: A breakdown of the error types.

4.4 Summary

We present the first model that provides high accuracy in both event and participant parsing. The model adopts a hierarchical design to model both the sequence of tokens and the sequence of script events and participants. Further exploitation of domain adaptation and data augmentation methods yields a substantial performance boost. This work has established methods to accurately parse both script events and participants, in a supervised learning framework. In the next chapter, we present our approach to script parsing on unseen scenarios, which is to see itself benefiting from the insights and utilities made possible by the supervised parser. A script parser on unseen scenarios potentially lifts the requirement on finely-annotated data, thus enabling wide-coverage script acquisition.

Chapter 5

Zero-shot Script Parsing

In chapters 2 and 3 we see how script knowledge benefits NLP applications. However, script knowledge could only be useful if annotations are available for the dataset. These annotations are expensive, not only because they are manual, but also because of the need for a large input on the expert level (see Modi et al., 2016, ; see also Section 2.3).

In this chapter, we automate the human annotation process, which reduces the cost of script acquisition, potentially alleviating the coverage issue. The solution is a 0-shot script parser, which takes unannotated texts on an unknown scenario as input and outputs the set of events, the set of participants for that scenario. The parser also clusters event and participant evoking tokens which indicate whether they evoke the same event or participant. Equivalently, the event and participant evoking tokens are now labeled.

The performance of our system is on par with that of a previous supervised model, which shows its potential as a substitute for manual annotations.

5.1 Introduction

The Coverage Issue

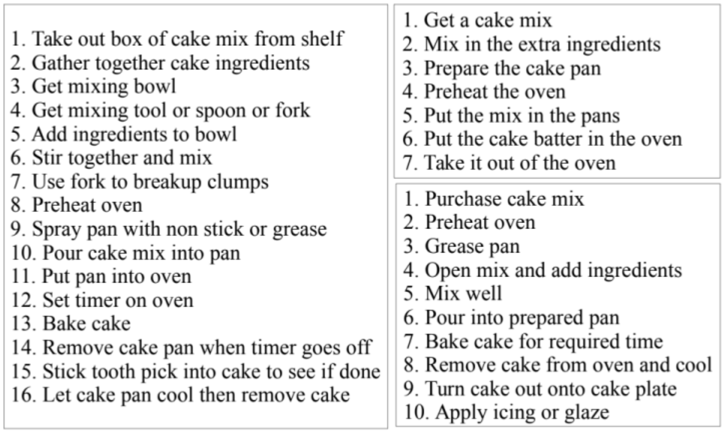
The acquisition of script knowledge is expensive. Spelling out the prototypical events and participants of a scenario might not seem difficult at first glance, as one who understands the task could retrieve it from his memory and spell it out. However, different people give different answers, and all versions of the scenario could be constructed into the production of natural language. It is highly nontrivial to formalize these different descriptions into a dataset which is accessible to machines.

Finding out how script events and participants can be expressed in surface text is also nontrivial. Each event and participant could be realized in surface language in various different ways. From the perspective of building a corpus, without further inductive bias, we would need a sufficient number of parallel text evoking each event and participant to encode the correspondence between script knowledge and surface text.

Existing corpora such as InScript and DeScript require heavy manual input. For DeScript, we need to collect multiple event sequence descriptions for each scenario. These event sequence descriptions are of different lengths and see quite some variations. In order to induce event clusters, the authors of DeScript used a multistage semi-supervised learning method that includes careful seed construction and feature design. These steps require a considerable amount of manual input from the researchers themselves (see Figure 5.1; see also Wanzare et al. (2017)).

InScript collects around 100 stories for each of its 10 scenarios, to bridge script knowledge and surface language. These stories need to be annotated token-by-token to decide whether an event or a participant is evoked through a multiphase pipeline. The complex process incurs high demand for manual input. For example, the set of events and participants need to be established in advance before the actual annotation could begin. These scenario-specific sets, termed *templates*, were created by experts and iterate during the annotation process to address new events or participants that were not taken into account. Therefore, the need for manual input at the expert level is positively related to the number of scenarios, which means that the pipeline is not feasible at large scales.

Existing resources of script knowledge only have very limited coverage due to their high costs. But script knowledge cannot be of practical usefulness without wider coverage. In this chapter, we approach the coverage issue by automating the process



1. Take out box of cake mix from shelf
2. Gather together cake ingredients
3. Get mixing bowl
4. Get mixing tool or spoon or fork
5. Add ingredients to bowl
6. Stir together and mix
7. Use fork to breakup clumps
8. Preheat oven
9. Spray pan with non stick or grease
10. Pour cake mix into pan
11. Put pan into oven
12. Set timer on oven
13. Bake cake
14. Remove cake pan when timer goes off
15. Stick tooth pick into cake to see if done
16. Let cake pan cool then remove cake

1. Get a cake mix
2. Mix in the extra ingredients
3. Prepare the cake pan
4. Preheat the oven
5. Put the mix in the pans
6. Put the cake batter in the oven
7. Take it out of the oven

1. Purchase cake mix
2. Preheat oven
3. Grease pan
4. Open mix and add ingredients
5. Mix well
6. Pour into prepared pan
7. Bake cake for required time
8. Remove cake from oven and cool
9. Turn cake out onto cake plate
10. Apply icing or glaze

Figure 5.1: Event sequence descriptions of baking a cake from DeScript. These event sequence descriptions are of different lengths. Inducing event clusters from these ESDs is a challenging task.

of annotating InScript.

Approach

Automating the annotations of events and participants in InScript presents us with the task of script parsing. There are different settings for the problem. Most interestingly, the availability of scenario-specific supervision. If annotated data are available for the test scenario, we have the task of *supervised script parsing*. We have shown in the previous chapter that supervised script parsing could be performed accurately with a hierarchical sequence model. However, these settings do not hold in the situation where we wish to expand the coverage of script resources because for the target scenarios (potentially the test set), we do not have annotated data. Namely, we need to perform inference on *unseen* scenarios.

Figure 5.2 provides an illustration of the task. Essentially, it is a clustering task where we need to decide which candidates evoke the same event or participant. Meanwhile, the set of clusters corresponds to the set of events or participants.

The picture is slightly different with the availability of InScript. Although InScript does not include any test scenarios, we believe that it packs knowledge about how script parsing is performed in general. This makes it a *zero-shot learning* task.

The key idea of our approach is to learn appropriate contextualized word representations to accommodate script parsing on an unseen scenario as a clustering task. We

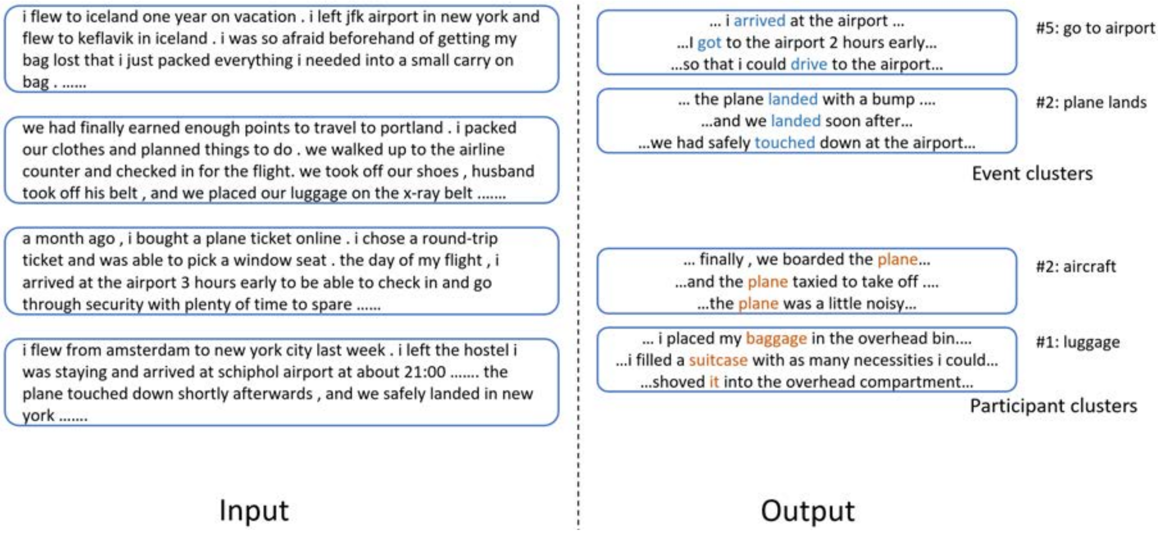


Figure 5.2: An illustration of the task. Given multiple narratives on an unseen scenario, we are tasked to produce the set of events, the set of participants and to label the original text with these tags. Equivalently, the candidates in the original texts should be clustered according to the event or participant they evoke.

begin with general-purpose word embeddings, which, we believe, encode features that are informative to script parsing. Such an embedding might also encode features that are not relevant to script parsing. We train a transformation which suppresses irrelevant features, whereas promotes informative features to produce a script parsing specific embedding. This approach is feasible because annotated corpora like InScript present us with how script parsing is performed. During the inference phase, the transformed embeddings are then fed into a clustering process to yield the results. In order to train the transformation, we define a consistency measure between an embedding and an annotated corpus, to quantify how much the embedding function fits the annotations. The consistency is a differentiable function that considers the similarity of the embeddings of candidates that are in the same cluster according to the annotations and the dissimilarity of embeddings of candidates from different clusters. We optimize this consistency to acquire better representations.

Our work is the first known attempt at performing script parsing on unseen scenarios. The model reaches a micro-F1 score on zero-shot event labeling of up to 68.1 and on participant labeling of up to 74.4, on par with the supervised model of Ostermann et al. (2017) that assumes training data for the same scenario. Qualitatively, when executed on texts similar to the training data, temporal script graphs of good quality could be extracted from the output of our parser; these automatically generated annotations could also accommodate tasks like story generation. We also find in probing tasks that our model learns to amplify information about sentence ordering from the pretrained embeddings, while suppressing low-level information about morphology and syntax, which are less relevant for the script parsing task.

In the next chapter, we apply the parser to unseen scenarios from a few different corpora and evaluate the quality of the outcoming script knowledge.

5.2 Task

5.2.1 Zero-shot learning

Zero-shot learning is a category of problems in machine learning. Under the zero-shot learning settings, no supervision is available for the test classes. That means, during the test time, the classifier receives samples from classes that are not observed during training and need to predict which classes they belong to. These test classes are referred to as *unseen* classes.

Zero-shot learning is proposed by Chang et al. (2008), where it was called *dataless classification*, in reference to the fact that no annotated data is needed for the test classes. The term *zero-shot* was first seen in Lampert et al. (2009), which refers to the fact that *zero* labeled data point is available for each test class, in contrast to *one-shot* learning and *few-shot* learning.

Zero-shot learning is a special case of domain adaptation. It might seem miraculous that a classifier could learn to predict samples from classes that do not offer any training signal. However, zero-shot learning is only feasible when the training data distribution and the test distribution admit sufficient mutual information. That means, the training data and the test data are similar to each other, from some certain point of view. For example, we would anticipate learning to identify mountain names helps learning to identify river names, but it does not offer as much help as it might do script parsing.

Zero-shot learning is a feasible scenario for various NLP applications. Consider any task for which a large body of homogeneous texts is available, yet only a small fragment is annotated. For tasks like event extraction, topic identification, etc., it is well conceivable that the unannotated texts include new classes that do not appear in the annotated part. If a model is trained on the annotated part and performs inference on the unannotated part, then a zero-shot setting is complete. As another example, consider a slightly different setting where texts from some low-resource language are not annotated, whereas those from the high-resource language have annotations. Now the annotated and unannotated texts are not homogeneous but still have positive mutual information: they are all, most probably, generated by homo sapiens that are not completely insane for the purpose of communication; these texts are all supposed to be consistent with the common sense knowledge associated with the same physical world. Although they might have different sets of labels for differences between languages, positive mutual information is still evident.

One common approach to zero-shot learning is through representation learning. In particular, we learn a latent representation space into which all instances embed. The representation could be trained with the training data. In classification tasks, one possibility is to optimize the *similarities* among training instances that receive the same label. As a result, over the course of optimization, the knowledge of the training data is encoded in the embedding function and could be transferred to the test data. For example, clustering test data in this embedding space might end up with clusters that admit similar structures as training classes. Zero-shot learning has been used to tackle data sparsity in various situations, such as machine translation for low-resource languages (e.g. Pham et al., 2019; Zhang et al., 2020; Johnson et al., 2017), generation (Duan et al., 2019; Philip et al., 2020), text classification (see, e.g. Yin et al., 2019) and question answering (e.g. Banerjee and Baral, 2020).

5.2.2 Script Parsing as a Zero-shot Learning Task

Our parser is tasked with predicting event and participant annotations for scenarios that are not seen during its training. Equivalently, this is a clustering process of potential event candidates and participant candidates. After the clustering is completed, the cluster indices will serve as class labels.

The structure of our zero-shot script parser is as follows.

The training is about learning a transformation φ that maps pretrained, contextualized, general-purpose word embeddings into a representation space that is more suitable for script parsing. More specifically, the embeddings of candidates that correspond to the same event or participant classes should be *more similar* than those that do not. The representation space has two sources of knowledge: the general-purpose word embedding and the training data. The former is of interest, as it is often the case that candidates evoking the same event or participant are similar in the sense of basic semantics. For example, both *Yesterday I went grocery shopping* and *Yesterday I went to the grocery store* could evoke a go to grocery store event; all mentions of the protagonist are probably *I* or *me*. But this is not always the case. One could also say *I drove to the grocery store*, where *drove* is not very similar to *go*; we could also find *I had to do grocery shopping*, which is a bit different. The participant groceries could be *eggs, vegetables*, could be *meat*, or *all I needed for a picnic*. A general-purpose word embedding does not consider these instances similar. This aspect of the embedding space is learned from the annotated training data, from which we know which instances belong to the same class, so we could optimize the transformation accordingly.

In the inference time, the first step is to identify both event candidates and participant candidates from the input text. This could be done according to shallow linguistic features. That is, all verbs are selected as event candidates and NP heads as participant candidates. The candidates are represented by a single token, but include its context as well.

Afterwards, we take the representations of all candidates from all input texts in the target scenario and apply φ to transform these representations to acquire the representation we learned during the training phase. Now, we cluster these candidates into sets of events and participants accordingly. The indices of these sets represent the event and participant labels of the unseen scenario.

The task is based on InScript, which consists of 10 scenarios. We take the texts from eight scenarios as the training data, one for validation and one as the test scenario, which means, the validation scenario and the test scenario are unseen in the training phase. During inference, the model takes the unannotated stories of the test scenario as input and labels them with events and participants types.

5.3 Model

In this section we elaborate on the model design. We address its training behaviors and inference behaviors separately, as they are quite different: training is about learning a representation according to the training data, whereas inference is mostly about executing clustering based on the learned representation.

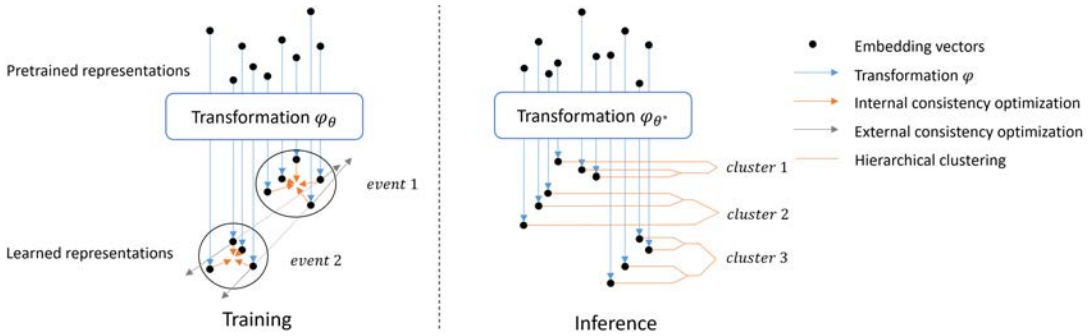


Figure 5.3: The overall framework. For the training, we learn the transformation according to the training data in the sense that the classes in the training set form better clusters in the new representation space. During the inference phase, we execute agglomerative clustering on the candidates. The regularizors are not depicted.

In the training phase, we run XLNet (Yang et al., 2019), parameters frozen, on the input text, to obtain contextualized word embeddings $f(c)$ for each event and participant candidate c. Later on, we train the transformation φ to minimize the distances within the same event and participant classes and maximize those between different ones (Section 5.3.1); the objective function also contains a few regularization methods that make use of coreference information, dependency information, and a more script parsing specific representation instead of raw XLNet.

In the inference phase, given the input texts on the test scenario and specifications of the event and participant candidates in it (this is acquired during data preprocessing parameter-free, see Section 4.1), the first step is to identify the candidates that evoke regular events or participants, to exclude irregular candidates from the clustering process. Afterwards, we apply the learned representation on these candidates and perform agglomerative clustering on them. The overall framework is illustrated in Figure 5.3.

5.3.1 The Objective Function

Now we introduce the training objective with which we learn the transformation φ. The purpose of our representation learning, in general, is to map instances from the same class to similar representations; similarly, those not in the same class should not be too similar. These intuitions are quantified by our *consistency* objective, which meansures how much a representation fits the class specifications in the given annotated data.

External Consistency

Let $\mathcal{C}$ be the set of all event candidates or the set of all participant candidates in the narratives associated with any scenario, and let $\pi(\mathcal{C})$ be a partition of $\mathcal{C}$. The partition defines an equivalent relation according to the training labels: each element of $\mathcal{C}$ consists of all events or participant instances that receive the same label. The notion could also be used for other equivalent classes of candidates. For example, *belong to the same coreference chain* is an equivalent relation over all participant candidates. We write $\pi(c)$ for the element of the partition to which the candidate c belongs. Given a pretrained embedding function f and the parameterized transformation φ_θ that we want to learn, we consider the average distance between instances belonging to different clusters:

$$d_{ext}(\pi(\mathcal{C});\theta) = \underset{\substack{c,c' \in \mathcal{C}:\\ \pi(c) \neq \pi(c')}}{mean} d(\varphi_\theta(f(c), \varphi_\theta(f(c')))$$

Here f is a pretrained XLNet encoder. There are various options in terms of the implementation of the transformation φ, for example attaching an additional dense layer or transformer layer to the output end of the encoder. Empirically, fine-tuning the last layer of the encoder yields the best performance. Moreover, it also implies more stable optimization than training a module from scratch.

We would like to push the embeddings of two candidates from different classes apart if they are *too close* to each other. We do so by maximizing a relaxed version of d_{ext}, the *external consistency* of the partition π:

$$\gamma_{ext}(\pi(\mathcal{C});\theta) = \underset{\substack{c,c' : \pi(c) \neq \pi(c'),\\ d(\varphi_\theta(f(c), \varphi_\theta(f(c')))\\ < \sigma_1 d_{ext}(\pi(\mathcal{C});\theta)}}{mean} d(\varphi_\theta(f(c), \varphi_\theta(f(c')))$$

Here $\sigma_1 \in (0, 1)$ is a hard threshold that quantifies *too close.* Intuitively, for the purpose of script parsing, it is not wise to further distance a pair of candidates from different clusters if they are already distant from each other. With the threshold σ_1, candidate pairs from different clusters are prohibited from generating a training signal if they are sufficiently distant from each other. The external consistency term captures the intuition that φ should map candidates from different classes to dissimilar vectors.

Internal Consistency

Likewise, consider the average distances between the embeddings of candidates from same classes:

$$d_{int}(\pi(\mathcal{C});\theta) = \underset{c,c':\pi(c)=\pi(c')}{mean} d(\varphi_\theta(f(c), \varphi_\theta(f(c')))$$

We would like to pull the embeddings of two candidates from the same class towards each other if they are too far away. In a similar spirit, we maximize *internal consistency* of π:

$$\gamma_{int}(\pi(\mathcal{C});\theta) = 1 - \underset{\substack{c,c' : \pi(c) = \pi(c'), \\ d(\varphi_\theta(f(c), \varphi_\theta(f(c'))) \\ > \sigma_2 d_{int}(\pi;\theta)}}{mean} d(\varphi_\theta(f(c), \varphi_\theta(f(c'))) \tag{5.1}$$

Similarly, the representations of candidates in the same cluster should not collapse to a single point. To this end, we implement σ_2, a hard threshold that draws the boundary of being too far away, which excludes candidate pairs in the same cluster from the generation of the training signal if they are already similar to each other.

We obtain an overall consistency measure γ, which we maximize in training; λ_i is a hyperparameter that balances the terms.

$$\gamma(\pi(\mathcal{C});\theta) = \gamma_{ext}(\pi(\mathcal{C});\theta) + \lambda_i \gamma_{int}(\pi(\mathcal{C});\theta) \tag{5.2}$$

this function could be seen as a variant of the triplet loss introduced by (Dong and Shen, 2018), but relaxed with the thresholds and uses a different distance metric. Empirically, these measures improve the stability of training and result in a moderate performance improvement.

The Dis-similarity Measure

We write $d(\cdot,\cdot)$ for the distance function in the representation space. Empirically, the following variant of cosine distance works well:

$$d(v,w) = \sqrt{1 - cos(\angle(v,w))} \tag{5.3}$$

Here $\angle(v,w)$ denotes the angle between v and w.

There are multiple possible choices for the distance function. The consideration behind this choice is a combination of implementation difficulty and performance.

To perform clustering, we use the implementation in *sklearn* (Pedregosa et al., 2011). The library is optimized to evaluate the Euclidean distance. In fact, given the scale of our problem (around 3k datapoints, each of 768 dimensions), pairwise distance evaluation easily goes intractable if what we use is noticeably more complicated than Enclidean distance. However, Euclidean distance per se is not a good choice as it is unbounded, which renders the training unstable. Most importantly, Euclidean distance does not deliver good performance after all.

Our design of d is tractable and also delivers good performance. Our distance function

$$d(v,w) = \sqrt{1 - cos(\angle(v,w))}$$

actually equals to

$$\frac{1}{2}\left\| \frac{v}{\|v\|_2} - \frac{w}{\|w\|_2} \right\|_2$$

which means it could be calculated by taking the Euclidean distance between the unit vectors corresponding to v and u, respectively. Therefore, in practice, the only operation we need to take apart from taking the Euclidean distance is normalizing the vectors to their unit vectors, which is cheap. Therefore, our formalization retains the benefit from the optimization efforts for calculating the Euclidean distance and executes in an affordable amount of time.

What may raise one more concern is the fact that d is not well defined as a distance function in $\mathbb{R}^n$, because if two vectors correspond to the same unit vector, their *distance* would be zero, which is not positive-definite. In practice, however, this is never the case, and we could relieve the concern.

5.3.2 Regularization

The training objective as defined in Equation 5.2 could be further regularized to improve its performance. Here, we introduce these regularization methods. Two of these add extra terms to the objective, whereas the third uses word embeddings that are specialized for script parsing.

Coreference

Coreference refers to linguistic phenomena where multiple noun phrases have the same interpretation, that is, when they refer to the same referent. For example, every usage of a pronoun refers to the same referent as its antecedent. In a segment of text, all referring expressions to the same referent appear in linear order to form

(a) There is a bus stop down the street from my house . If you take it going south , it leads to the city...

(b) ...$\text{I}_{passenger}$ fed my coins_{money} into the slot where you put your money...
...$\text{I}_{passenger}$ boarded the bus and paid for my ride with my change_{money}...

(c) ...the bus $\text{arrived}_{bus\_stops}$ at the bus stop closest to the beach...
...I would need the bus to stop_{bus_stops} next to the hospital...

Figure 5.4: Examples from the taking a bus scenario. (a) coreference chain. (b) instances of event (pay) sharing similar participants. (c) instances of participant (destination) that have similar event dependents.

a *coreference chain*.

In computational linguistics, *coreference resolution* is a well-studied task which finds out all coreference chains in a text, or equivalently, determines whether any pair of referring expressions refer to the same referent. It could be performed accurately with out-of-the-box tools like Spacy (Honnibal and Montani, 2017, see). Although there are complex cases like those fitting the Winograd schema, the performance is usually above 90 points F1 score for corpora consisting of less complex texts, which is expected from InScript.

From the perspective of script acquisition, an immediate observation is that all referring expressions in the same coreference chain must evoke the same participant, be it regular or not. For example, all mentions of a particular bus in scenario taking a bus (see Figure 5.4 a) evoke the participant bus. Therefore, ideally, these noun phrases should belong to the same participant cluster and have similar representations.

We capture this intuition to integrate it into the training objective as follows. We denote the set of all participant candidates as $\mathcal{C}^c$. Let $\eta(\mathcal{C}^c)$ be the set of all coreference chains that consists of the participant candidates in $\mathcal{C}^c$. Therefore, $\eta(\mathcal{C}^c)$ is a partition of $\mathcal{C}^c$. Like π above, $\eta(\mathcal{C}^c)$ also specifies an equivalence relation, in that two candidates are in the same class if, and only if, they are in the same coreference chain. Now, if we revisit the notion of consistency we have, we could observe the analogy between coreference chains and participant classes. Each of them defines a partition of $\mathcal{C}^c$; we would like our representation for script parsing to consider the items in the same part similar, which applies to both partitions; similarly, items in different parts of each partition should have different representations. These observations inspire us to formulate a coreference chain consistency measure with the *internal consistency* we defined, i.e. as

$$\beta(\theta) := \gamma_{int}(\eta(\mathcal{C}^c); \theta)$$

In the training phase, we add this term to the objective function as a regularizor.

Event-participant dependencies

The second regularizor is based on dependencies. Events and participants in a script are closely related: In multiple narratives in the same scenario, if two verbs have arguments from the same participant class, they tend to describe the same event (Figure 5.4b); and if two noun phrases are arguments of the same event, they tend to describe the same participant. As an example, consider taking a bus, where bus is a participant. Now, there are only a proportion of events, whose evoking verbs could possibly take the bus as a subject or an object (or whatever other semantic roles). The bus could arrive at destination, start, could be boarded, but cannot be paid. It is the ticket that is supposed to be paid for. Thus, the potential event or participant classes of a dependent are informative to script parsing (see also Figure 5.4c).

Once again we observe the analogy between what we want to abstract here and candidates who receive the same event or participant label. We introduce a regularizor with regard to dependency relations as follows. Let c_p be the set of event candidates that have participant p as an argument; we encourage φ to map the elements of c_p to similar representations. Let $\xi(\mathcal{C}_e^d)$ be the set of all c_p, that is, a partition of event candidates based on dependencies. Analogously, let $\xi(\mathcal{C}_p^d)$ be the set of participant candidate sets that depend on the same events. We can formulate a dependency-based consistency measure as

$$\alpha(\theta) = \gamma_{int}(\xi(\mathcal{C}_e^d); \theta) + \gamma_{int}(\xi(\mathcal{C}_p^d); \theta)$$

The final training objective, with hyperparameters λ_c, λ_d and cluster assignments $\pi^*(\mathcal{C}^e)$ of event candidates and $\pi^*(\mathcal{C}^p)$ of participant candidates as is specified by the annotations in InScript, is

$$\theta^* = \underset{\theta}{argmax}[\gamma(\pi^*(\mathcal{C}_e^d); \theta) + \lambda_p \gamma(\pi^*(\mathcal{C}_p^d); \theta) + \lambda_c \beta(\theta) + \lambda_d \alpha(\theta)] \tag{5.4}$$

Specialized word embeddings

Finally, we improve our model by using more specialized word embeddings as input instead of general-purpose XLNet embeddings. These specialized word embeddings come from the training of a supervised script parser, which is going to be the model we introduced in Chapter 4: as a side product of optimizing the model, we have an encoder that produces word embeddings that deliver good results for script parsing. Our hypothesis is that the training of the supervised model should promote features that are more relevant to script parsing. As a result, the outcoming representation should be a better starting point than the general-purpose XLNet from which we train a representation for zero-shot script parsing.

We acquire the specialized word embeddings by retraining the model in Chapter 4 with our current training data. We cannot use the model directly, as the training data of the supervised parser overlap with the test set of our zero-shot parser. Therefore, we revisit the supervised parser training process to implement the necessary adjustments and use it as a feature extractor.

The optimization of the feature extractor is different from that of Chapter 4 in the following ways. First, we designate its training data and validation data by performing a random split of the current training set, that is, the eight scenarios we kept for training the zero-shot script parser. Second, the model is trained on regular events and participants only, as irregular ones are excluded from the training for our zero-shot parser. We replace the embeddings f in the consistency formulars with the supervised parser (excluding the final classifier) as the initial representation. Finally, we use a different set of hyperparameters.

5.3.3 Inference

Whereas the training of our parser is about representation learning, inference is about clustering candidates according to the representation we acquire during training. In particular, we firstly identify *regular* candidates; secondly, we execute agglomerative clustering on these candidates to produce the event and participant clusters.

Regular candidate identification

Script parsing is focused on *regular* candidates, because irregular candidates are not closely related to the scenario and are not always specific to the scenario. They are not of interest to script acquisition, as we are after the prototypical manner that script scenarios evolve, including the set of events, participants, their dispositions over a narrative, etc. The random incidents or entities that happen to be around in some instantiations of the scenario are not our target. Technically, irregular

Input Data points $\mathcal{C}$; threshold $\rho \in R_{\geq 0}$; cluster dis-similarity function $d : \mathcal{P}(\mathcal{C})^2 \to R_{\geq 0}$.

Output A partition $\pi(\mathcal{C})$ of $\mathcal{C}$.

Steps

1. Initiate the set of clusters $\pi_0(\mathcal{C}) \leftarrow \{\{c\} : c \in \mathcal{C}\}$.
2. If $\pi_0(\mathcal{C})$ is a singleton, terminate and output $\pi_0(\mathcal{C})$.
3. Get $c_1, c_2 \in \pi_0(\mathcal{C})$ that minimizes $\mathbf{d}(c_1, c_2)$.
4. If $d(c_1, c_2) > \rho$, terminate and output $\pi_0(\mathcal{C})$. Otherwise, merge c_1, c_2: $\pi_0(\mathcal{C}) \leftarrow (\pi_0\mathcal{C} - \{c_1, c_2\}) \cup \{c_1, c_2\}$ and go to step 2.

Figure 5.5: The clustering algorithm. Its time complexity is $\mathcal{O}(n^3)$. We use the out-of-the-shelf implementation from **sklearn** (Pedregosa et al., 2011). The threshold ρ is estimated from the training data.

candidates are an ill-formed class in terms of clustering: irregular candidates make a diverse group of instances without much semantic similarity to each other; they usually do not refer to the same action or entity, in contrast to regular events and participants. That means establishing a representation to group together irregular candidates would require very different features than that for regular classes. Including them in the training of the representation space would incur additional complexity to the problem and deteriorate the performance. Therefore, irregular candidates are excluded from the training. Note that the exclusion of irregular candidates is still a part of our pipeline, thus relevant to the evaluation.

To this end, we train a classifier, the *regular candidate identifier*, to distinguish regular and irregular candidates, in order to exclude the latter from representation learning. The classifier is essentially the model for supervised script parsing that we present in the previous chapter, with the exception that we collapse the output classes, so the model only distinguishes regular candidates from irregular ones. The set of output labels now consists of REGULAR_EVENT, IRREGULAR_EVENT, REGULAR_PARTICIPANT and IRREGULAR_PARTICIPANT. The training pipeline mostly stays the same, except that we use a different training split, which is restricted to the eight scenarios we reserved for training. The hyperparameters were changed as well. The rest of the optimization is identical to that of the model in Chapter 4. The performance of the classifier is around 85 points F1 score.

Clustering

With the irregular candidates excluded, we feed the regular candidates to the transformed representation we learned, to acquire the representations of these candidates. Now, we have the embeddings $\varphi_{\theta^*}(f(c))$ for each candidate c, and we are ready to cluster them to obtain the sets of events and participants.

The clustering algorithm that we use is agglomerative clustering, which is a hierarchical top-up clustering algorithm(see, e.g. Lukasová, 1979). Starting with each data point considered a singleton cluster, the algorithm iteratively merges the most similar pair of clusters. It terminates when the stopping criterion is met. There are two possible stopping criteria, namely, to stop when the number of clusters decreases to a predefined quantity or when the minimum dissimilarity between the current clusters goes beyond a predefined threshold, which means that the current clusters are considered too *dis-similar* to be merged. The time complexity of the agglomerative clustering is $\mathcal{O}(n^3)$, which is tractable given the scale of our data. See Figure 5.5 for the algorithm.

In our case, the data points are the embedding vectors, thus $\mathcal{C} \subseteq \mathbb{R}^n$. For the stopping criterion, since the number of event and participant clusters is assumed to be unknown, and they vary among different scenarios, it is not possible to set a minimum number of clusters as the stopping criterion. Therefore, we use the dissimilarity threshold stopping criterion. The threshold is estimated from the training data. More specifically, for each scenario, we take a quantity which allows the clustering process to come up with exactly the same number of clusters as is annotated. We then averaged the 8 training scenarios to get the estimation. The dissimilarity measure could also take multiple forms, such as average Euclidean distance, minimum Euclidean distance, etc. Empirically, the variance of the union turns out to be the best measure of dissimilarity.

The Protagonists

We treat the protagonist differently from the other participants during the inference phase. The protagonist is a special participant present in every scenario, which links most of the narrative event chains. It also participates in most events belonging to the scenario. For example, the passenger in taking a train, the customer in grocery shopping or the rider in fixing a flat tire. In narratives of personal experiences, the protagonist is the most frequent participant in all scenarios; the referring expressions to the protagonist always forms by far the largest class of participants.

From the perspective of participant parsing, tokens evoking the protagonist are easy to detect in narrative texts. Referring expressions to the protagonist always forms by far the longest coreference chain. Moreover, the vast majority of these referring expressions are actually first personal pronouns, such as *I, me, we, us, our, ours* etc. We have good reasons to believe that a simple, parameter-free rule-based method could do a satisfying job in identifying protagonist mentions in the text. Therefore, it is unnecessary to include them in the complex neural pipeline. Instead, what we do is simply exclude them from the optimization (that is, consider these tokens unannotated). During the inference phase, we group every item in the longest coreference chain together as the protagonist cluster. This simple heuristics already yields more than 98 points of F1 score on the test set, which is sufficiently good. In fact, most of the errors result from the errors of the coreference resolution tool.

5.3.4 Implementation and Optimization

The model is implemented with AllenNLP 1.2 Gardner et al. (2017). The pretrained XLNet model we used is *xlnet-base-cased* (`https://github.com/zihangdai/xlnet/`). Training is further regularized with weight decay. The optimization is performed with adam (Kingma and Ba, 2015) in conjunction with early stopping which monitors validation loss; the hyperparameter tuning is performed with random hyperparameter search (Bergstra and Bengio, 2012). Optimization takes on average 5 hours on a single Tesla v100. The implementations of agglomerative clustering and the Hungarian algorithm (which is necessary for evaluation) are from the *scipy* library.

5.4 Evaluation

5.4.1 Metric

The evaluation of our approach is not trivial. What we get as output of the parser is a number of event clusters and participant clusters of contextualized tokens. According to the annotation in the test set, we know the annotation these tokens receive, i.e., what event or participant they actually evoke, or do they evoke any event or participant at all. For some intuition, Table 5.1 gives a few sample event clusters and the ground truth, which is the annotation these tokens receive.

Usually, each output cluster would include evoking tokens for multiple events. Most of the time, there is a clear majority. Therefore, from a certain point of view, this

Cluster 1

[plane] ... finally , we boarded the plane...
[plane] ...and the plane taxied to take off ....
[plane] ...I took my seat in my aircraft...
[plane] ...I flew on an airplane about 2 month ago...

Cluster 2

[cabin luggage] ... i placed my baggage in the overhead bin....
[check in luggage] ...i filled a suitcase with as many necessities i could...
[cabin luggage] ...shoved it into the overhead compartment...
[cabin luggage] ...put the small bag I had under the seat...
[check in luggage] ...she asks me how many suitcase I am bringing...

Cluster 3

[check in luggage] ... i brought my ID and suitcase to the airport...
[check in luggage] ...she checked in my suitcase and tole me my gate...
[cabin luggage] ...the security scanned my purse...

Cluster	plane	Checkin lug.	Cabin lug.
#1	4	0	0
#2	0	2	3
#3	0	2	1

Cluster 1→ plane
Cluster 2→ cabin luggage
Cluster 3→ check in luggage

Participant	Accuracy	F1
plane	1.00	1.00
cabin luggage	0.60	0.67
check in luggage	0.67	.0.57
average	0.76	0.75

Figure 5.6: A set of example output participant clusters, the linear assignment problem it corresponds to, its solution and the outcoming metrics.

cluster should correspond to that majority event. It is more complex than it may sound like, as we occasionally have multiple clusters with the same majority label. Actually, the result of clustering distinguishes the clusters with mere indices like $cluster_1, cluster_2$ instead of which event it preferably corresponds to; moreover, it is not guaranteed that the number of clusters is exactly the same as the number of gold labels. What we wish to understand from the evaluation is how well the clusters produced by our parser align with the gold clusters. We assume that we have no knowledge of the inference scenario, including the set of classes. A decent evaluation metric would need to answer two questions: is the set of event and that of the participant correct, and whether the parser correctly maps all regular candidates to the event or participant they evoke.

To answer these questions, the first step is to establish the correspondence between the output clusters and the ground-truth classes. If we could map the clusters to the gold classes, we would be able to evaluate the *accuracy* of each image class as if it were a classification task. There are several preferences in terms of how the mapping should be established. As the number of gold classes is unknown, the mapping needs to be injective. Furthermore, mapping a cluster to a gold class seems *desirable* if they overlap a lot. We would like to quantify exactly how desirable a mapping would be from a global point of view. The most intuitive quantity is the number of instances that is mapped to its ground-truth label. For example, if we map a cluster of participant candidates to the participant class bus, we are interested in how many candidates there do evoke the bus participant. If we consider this number as the utility of the mapping, we could summarize the utility of each pointwise mapping as a matrix M s.t.

$$M_{ij} = \text{count of instances in the } i\text{-th cluster that belongs to gold class } j$$

Note we could always add rows or columns that consists of 0s to M to make it square, thus we assume, without lose of generality,that M is square. Observe that the total utility of the mapping defined by the main diagonal, which maps the i-th cluster to gold class i for each index i, is the trace of the matrix. The *optimal* mapping we are looking for, corresponds to a permutation of the matrix that maximizes its trace. Now we recognise the problem as an instance of the *linear assignment* problem. See Figure 5.6 for an example evaluation instance and its corresponding linear assignment problem.

The linear assignment problem is solved by the Hungarian algorithm (see, e.g. Kuhn, 1955) in cubic time, which is tractable given the scale of our problem (the size of M is less than 40×40). The algorithm outputs the optimal assignment, which maximized the overal instance wise *accueacy* induced by the assignment. With the optimal assignment, we could evaluate an F1 score based on it, which we term *Hungarian F1*, and use it as our main evaluation metric. This metric allows us to compare the results of clustering-based parsers with those of classification-based parsers. Figure 5.6 gives the solution to our example and the outcoming metrics.

Recall that InScript has 10 scenarios, 8 of which are randomly selected as training data, one for validation and one to test. We evaluate our model with 10-fold cross-validation. For the designing purposes of our parser, it is also interesting to see its performance on different corpora. We address this in Chapter 6.

5.4.2 Baselines

We evaluated several other models for comparison purposes. First, the model XLNet performs clustering directly according to the bare XLNet embeddings, rather than the transformed embeddings φ_{θ^*} we learned during the training phase. Word embeddings provide a decent basis for evaluating semantic similarities. An improvement over this baseline would validate our parser as a whole.

In addition to our *full* model, as specified by equation 5.4 with the specialized embeddings described in Section 5.3.2, we also present results for a few ablated versions, including those without the extensions regarding event-participant dependencies (*dep*), coreference (*coref*), and specialized embeddings (*specialized*).

Finally, we include the results of two supervised parsers, the one we presented in Chapter 4 (supervised) and the one from Ostermann et al. (2017). The gap between models that had scenario-specific supervision and those that do not is always interesting. However, it is worth a bit of our attention that these models are not strictly comparable to ours, as they receive a different data split for training, validation and test, due to the fundamental differences in their settings.

5.4.3 Results

For each clustering-based method, we report two results: one is acquired with access to gold regularity, while the other relies on the regular candidate identifier from Section 5.3.3. All variants use the same number of trials for hyper-parameter tuning (thus roughly the same computational resources). We also conduct five parallel experiments to test the models' robustness against random initializations, and report the mean and standard deviation.

ground truth	text
turn water on	... I might drain the tub and *put* in more water ...
sink into water	... I turn off the faucet and *sink* into bliss ...
sink into water	... Then I *slid* into the water and enjoyed the relaxing warmth for twenty or more minutes...
sink into water	... I gingerly *lowered* myself into the nice warm water and immediately began to relax...
sink into water	... I eased my way into the tub and let myself *sink* into the water ...
sink into water	... I slowly *sunk* the rest of my body , and closed my eyes...
sink into water	... the tub was full and ready . I *slipped* into the tub and soaked in the bliss...
washing tools	...then I lather up with either *soap* or shower gel ...
water	...After I scrub really good and finish singing , I pour *water* continuously on my body...
washing tools	...I pour water continuously on my body until all the *soap* was he s off...
washing tools	...I cleaned my hair with some *shampoo* and washed my body with a wash cloth and rinsed...
washing tools	...shampooed my hair and applied some *conditioner* then washed my body...
washing tools	...applied some condition er then washed my body using some *liquid body wash*...
washing tools	...After I have washed everything , I rinse the *soap* from my body with the water in the tub...
washing tools	...on the corner of the bathtub . I lather ed *it* up and washed my arms , my legs...
washing tools	...take a wash cloth and *soap* or body wash to give yourself a good scrub down...
washing tools	...You can put the *shampoo* in your hair...
washing tools	...place your head under the faucet to rinse out the *soap* . Enjoy your bath !
washing tools	...washed myself with a *wash cloth* and soap . Then I leaned my head against...
washing tools	...stepped into the bath tub . I used *soap* and a wash cloth to clean myself...

Table 5.1: Example output clusters. Top: event; bottom: participant. The table presents a random selection of instances from these clusters as the original output could contain hundreds of instances.

model	gold regularity	task	events		participants	
			macro F1	micro F1	macro F1	micro F1
Ostermann17	✓	supervised	58.1	66.0	n/a	n/a
supervised	X	supervised	75.1	85.7	80.3	90.3
XLNet	X	zero-shot	40.2	53.2	39.3	60.5
- dep, cor., spc.	X	zero-shot	$46.0_{\pm 2.8}$	$58.4_{\pm 2.7}$	$47.5_{\pm 2.6}$	$\mathbf{75.7}_{\pm 1.8}$
- dep, cor.	X	zero-shot	$48.6_{\pm 5.2}$	$62.7_{\pm 3.7}$	$44.5_{\pm 3.3}$	$71.8_{\pm 2.1}$
- dep	X	zero-shot	$51.0_{\pm 3.7}$	$66.8_{\pm 4.3}$	$\mathbf{52.0}_{\pm 3.7}$	$74.8_{\pm 2.9}$
Full	X	zero-shot	$\mathbf{53.4}_{\pm 1.8}$	$\mathbf{68.1}_{\pm 2.3}$	$51.7_{\pm 1.6}$	$74.4_{\pm 1.4}$
XLNet	✓	zero-shot	43.1	51.6	43.9	61.0
- dep, cor., spc.	✓	zero-shot	$46.1_{\pm 1.9}$	$55.4_{\pm 2.2}$	$51.1_{\pm 2.7}$	$\mathbf{75.3}_{\pm 1.3}$
- dep, cor.	✓	zero-shot	$55.3_{\pm 2.8}$	$65.8_{\pm 2.8}$	$52.5_{\pm 3.1}$	$73.6_{\pm 2.1}$
- dep	✓	zero-shot	$56.7_{\pm 3.3}$	$67.4_{\pm 3.7}$	$\mathbf{53.6}_{\pm 2.9}$	$74.2_{\pm 2.0}$
Full	✓	zero-shot	$\mathbf{57.6}_{\pm 1.3}$	$\mathbf{68.1}_{\pm 1.3}$	$52.8_{\pm 1.4}$	$73.7_{\pm 1.4}$

Table 5.2: Results. We show the average over ten-fold cross validation and five training runs when feasible. These quantities are the Hungarian versions of F1 defined in Section 5.4.1. Some models train and inference according to the regularity annotations in InScript, instead of the predictions of our regular candidate identifier. Ostermann and supervised use a data split where the models see the test scenario during training; the other variants use the zero-shot settings. Therefore, the numbers from Ostermann and supervised are not strictly comparable to that of the ones under the zero-shot learning settings. They are here for reference only.

The results are shown in Table 5.2. All variants of our model outperform raw XLNet embeddings by a considerable margin. Our model also performs on par with Ostermann's on event parsing, although we do not have access to scenario-specific supervision whereas Ostermann's does, and our model additionally performs participant parsing.

The model extensions significantly improve the performance of the parsing. Access to coreference information, as hypothesized, improves participant parsing performance. Dependency information also grants a performance boost in event parsing. A closer inspection shows that with dependency information, the parser is better at grouping together event candidates that contain different verbs, whereas the arguments of these verbs belong to the same participant class. For example, the event sink into water in taking a bath could be evoked by *slide into water*, *sink into water*, *slip into the tub*, *lower into the tub*, etc. All verbs in these event candidates share arguments *I* and *water* or *tub*, which our parser correctly clusters together. Without dependency information, the parser cannot handle such level of variation and mostly groups together candidates whose predicate is 'sink', the most frequent verbalization of the event.

In general, we obtain a higher micro-F1 for participants than for events, whereas the differences between the macro-F1s do not show a pattern as clear. There are two underlying causes: first, participant parsing per se is a less complex task from the perspective of micro-averaging, given the large, simple protagonist cluster; second, the distribution of participant class sizes is more skewed than that of the events, so the larger classes, for which performance tends to be better, take a higher proportion in the whole dataset.

The performance of our script parser differs from fold to fold: we get 70.1 micro-F1 for participant parsing in taking a bath, but only 43.8 on borrowing a book from library. This is anticipated, as these scenarios could be quite different in the number of events and participants, and more importantly, the patterns in their event chains. The choice of validation set might also be a factor, as the performances on different validation sets generalize differently to their respective test sets.

5.4.4 Error Analysis

As a side product of evaluating Hungarian F1s, we obtain the optimal assignment of output clusters to gold candidate classes, which equivalently labels each candidate with an event / participant class. A case is considered an error if this assignment differs from its original annotation. We manually inspect 30% of the test errors made by our full model with predicted regularity case by case and categorize them. We analyze these error types with examples from scenarios taking a train and Taking a bath.

Granularity

Many events could be divided into multiple subevents, forming a hierarchy of events. For example, in the taking a bath scenario, we have prepare for bath, undress and grab a towel. A similar phenomenon is observed for the participants. This fact manifested itself as various types of errors. For example, the set of event labels in InScript often consists of events of different granularities, frequently rendering multiple cluster assignments feasible (e.g. *… I* ***took*** *a clean towel with me …* in either prepare for bath or grab a towel). As a result, the parser sometimes confuses one event cluster with another that includes it or groups together different events that actually fit together (turn on water and fill tub with water). Granularity accounts for two-thirds of event errors and one-sixth of the participant errors.

Shared predicate or argument.

Some incorrectly clustered events share the verb or some arguments with another class, especially when light verbs are involved, making the distinction harder. For example, in taking a train, a few instances of the get ticket event (e.g., "I took the ticket from him") are predicted as conductor checks ticket (e.g., "I gave the ticket to him")[1].

Number of clusters

As the test scenario is unseen, our parser does not know how many clusters there should be, but rather terminates the clustering process with a similarity threshold. Therefore, we often end up having a different number of clusters than the corpus, which results in larger classes being split into more than one or multiple smaller classes being merged into one. For example, for taking a train, the model yields 14 event clusters, whereas there are 15 in the corpus. In such cases, the parser has to compromise, causing damage to its performance. As a result, no predicted cluster was assigned to the door opens event, which refers to the opening of the train door before passengers board. Instead, 5 out of its 6 instances end up in the same cluster that mostly comprises instances of the get on the train event, a fair compromise.

Scenario-specific knowledge

The primary component of hard cases comprises candidates that are only equivalent when conditioned on the scenario. For example, event *spend time in the train* collects activities like *purchasing a coffee*, *taking a nap*, *looking out of the window*, etc. Our parser can partially address these instances, as it has access to the entire event chain of the story (so it knows all these events happen between *take a seat* and *get off the train*), but the performance is weaker.

Reasonable Alternatives

Interestingly, the model produces some reasonable clusters, although it does not agree with the annotation. For example, there is a spend time in the train event, which collects mainly entertainment during the ride, such as eating, playing games,

---

[1] The dependency regularizor moderately magnifies this issue, as with it the parser tend to cluster candidates that share similar dependents together. But this is outweighed by the performance gain it provides

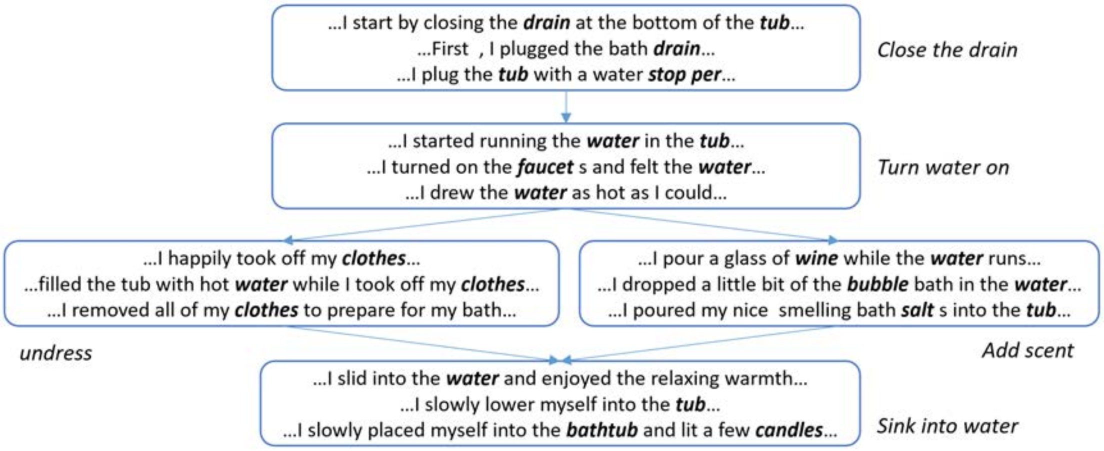


Figure 5.7: A fragment of the temporal script graph for taking a bath inferred from our parsing result. Each node illustrates 3 random verbalizations from the cluster. Gold classes are shown on the side. Further edges that could be inferred by transitivity are omitted. We see that one could first undress or add scent (to the bath tub) first before sink into water.

etc. A small fraction of this event is about ‘sitting, such as ... *i got to relax and* ***have*** *a compartment with complete silence, where I then took a nap...*, and *...I quietly* ***sat*** *by the window seat and looked at the great scenery outside...*. These candidates are clustered into the get to seat event. We also have a cluster that includes most instances from get to train station and get to the platform.

5.5 Further analysis

5.5.1 Temporal Script Graphs

Script knowledge could be neatly illustrated with temporal script graphs. Now we use the result of our zero-shot parser to compile these graphs for the unseen scenarios (i.e., the test set).
As a type of procedural knowledge, the *temporal order* among the events is a most important part of script knowledge. Some events must precede some other ones, for example, in the taking a bath scenario, one needs to turn water on before sinking into water. However, the temporal order is not a total order. For example, we could have undress first or add scent first.

We establish the temporal order as follows: we say event e_1 *precedes* event e_2 if and only if, in stories where they both occur, the proportion where e_1 occurs before e_2 is beyond a threshold $\zeta \in (0.5, 1)$. If neither e_1 precedes e_2 nor e_2 precedes e_1, we decide that they could follow an arbitrary order. Finally, we take the transitive closure of what we already have.

With the temporal order established, we are ready to compile the temporal script graphs, which depict the temporal order as a directed acyclic graph and collects different verbalizations of each event. See Figure 5.7 for an example.

As a means of evaluation, we view the construction of temporal script graphs as a task of retrieving temporally ordered event pairs and evaluate our results against those inferred from InScript annotations. What our clustering results yields admits 75 points F1 score.

5.5.2 The Script-specific Embedding Space

We conjectured that the transformation φ is learned to distill the relevant information for script parsing out of the general-purpose pretrained XLNet embeddings. One observation of script parsing is that the task is beyond the sentence level: the progress of the instantiation of the scenario contains key information for determining event and participant types. Lower-level features, like part of speech, are less relevant given that we have already identified the candidates in preprocessing. Now, we verify these points by comparing the representations we learned with the original XLNet embeddings.

What we do is probing both embeddings on a series of NLP tasks that focus on different semantics levels, to see if a shift of focus towards higher level semantics takes place during the training, as we believe the higher level is more important to script parsing. Our experiment uses four tasks. (1) POS tagging and (2) named entity recognition depend mostly on the token itself and its local context. (3) noun phrase chunking raises the level a bit higher, as it is determined by sentence-level syntax. (4) Sentence ordering is at the highest level. For this task, the classifier receives a story as input and needs to determine whether the sentences in the story appear in the original order or have been randomly shuffled. This decision would require information across the entire story to be made. (1)-(3) are formulated as sequence labeling tasks; (4) is a binary classification.

These experiments are conducted on InScript. The data split we use is the same as that for training the specialized embeddings (random split on the 8 training

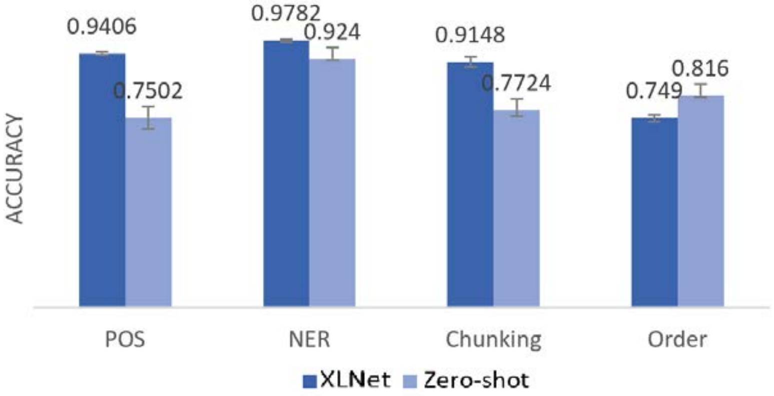


Figure 5.8: The performance on the probing tasks. The specialized representation is significantly better at sentence ordering, the task at the highest level of semantics, whereas its performance is weaker on the other tasks. The error bars show one standard deviation. All differences between these pairs are significant at $\alpha = 0.05$ according to independent T-test.

scenarios). As for the supervisions, we use POS as is annotated in InScript. For NER and chunking, the labels are generated with the larger model from Spacy (Honnibal and Montani, 2017, model *en_core_web_trf*).

We train a classifier for each task and each embedding space. All models take the simplest architecture, that is, the encoder followed by a fully connected layer. This is customary for probing experiments, as a trainable classifier with limited capacity guarantees that the behavior of the embeddings plays the primary role in the performance.

The parameters of the embeddings are all frozen. Hyper parameters are selected by random search. For each of these probing tasks, both representations use roughly the same amount of GPU budget.

The results are shown in Figure 5.8. The transformed representation $\varphi(f(c))$ incurs performance drops on most tasks, compared to general purpose embeddings $f(c)$. However, the performance on the sentence ordering task, the only one that requires information throughout the whole story to be performed, sees a noticeable improvement. This supports our hypothesis that φ amplifies higher-level features, which are more important to script parsing than to generic language modeling. In comparison, lower-level information about morphology and syntax is deemphasized.

5.6 Summary

In this chapter, we presented a zero-script parser that operates on unseen scenarios. The key design is to perform representation learning based on the scenarios for which annotation is available and to transfer the knowledge to unseen scenarios. Its performance on InScript is on par with that of a previous parser that had access to scenario-specific annotations. The parser we develop in this chapter allows us to acquire script knowledge from a number of unannotated narratives on a new scenario, which is significantly cheaper than manual annotation. In the next chapter, we apply this tool to different sources of unannotated texts and evaluate the quality of the script knowledge we are to acquire.

Chapter 6

Wider Coverage Script Knowledge

6.1 Introduction

The purpose of this thesis is to expand the coverage of script knowledge. With limited resources, that means performing script acquisition at a lower cost than that of InScript, which requires massive manual annotation.

We have already automated the annotation process in the previous chapter by developing our zero-shot script parser. The parser takes annotated training data, but it is able to perform script parsing on texts about completely unknown scenarios, which fits the needs of script acquisition on these scenarios. The parser was evaluated in InScript by leaving out the data of a scenario in its entirety as the test set, which creats an *unseen* scenario. The parser exhibits good performance in InScript. However, to evaluate its performance under more realistic script acquisition settings, we need to evaluate it on different texts.

In this chapter, we evaluate our zero-shot parser on corpora different from where it is trained. The corpora we use are MCScript and Spinn3r. MCScript is a question-answering dataset whose background texts are similar to InScript in the sense that they are stories about various script scenarios and that it consists of a relatively simple language. Spinn3r is a collection of personal blogs; these texts are naturally generated and more diverse. Table 6.1 gives some example stories from these corpora (see Chapter 2 for more details).

Our evaluation is two-fold. Intrinsic evaluation uses the same Hungarian F1 as defined in the previous chapter. For the extrinsic evaluation, we revisit the downstream task of story generation. In particular, we run our parser on MCScript as an annotation tool, then we train the story generation model as presented in Chapter 3 with the data annotated by the zero-shot parser. On MCScript, our parser delivers decent results by finding reasonable event and participant clusters and accommodating the training of the story generation model. On Spinn3r, the performance is not sufficient for script acquisition, which might be a combined consequence of both the amount of scenario-specific data available and the variety of language. In conclusion, we found the zero-shot parser a functional tool for script acquisition from MCScript style scenario descriptions. These texts might still need to be crowd-sourced, but the vast majority of its cost resulting from the complex annotation process could be lifted.

InScript
after playing football with my friends , i was all sweaty and dirty , so i decide i should take a bath . first i made sure there were clean towels available and enough soap and shampoo to thoroughly clean my skin and hair . i put the plug in the bottom of the bath tub and turned on the water , being careful to adjust the water to a nice warm , but no too hot , temperature . i used my finger to determine just the right temperature . after removing my clothes and placing them in the hamper , i carefully stepped into the tub , being sure to steady myself as i entered . i gingerly lowered myself into the nice warm water and immediately began to relax . after a few minutes of rest and relaxation , i got out the soap and thoroughly washed my whole body , making sure to get all of the dirt and grass stains off that i had accumulated from playing football . next i washed my hair with shampoo and laid back into the water to really rinse it out well . i carefully exited the bath tub , making sure to step onto the bath mat so i would n't slip on the smooth floor tiles , and got a towel and dried myself off . now my bath was complete and i felt so nice and clean ! what a great feeling !
MCScript
Mary am has always had her children help her make the picnic food . Now that they are older , she supervise s them more than she actually cook s . Molly boil s eggs to stuff , while Pol ly boil s potatoes for potato salad . none Terri makes a poppy - seed dressing for the fruit salad , and a white dressing for the potatoes . When the eggs are done , Molly cool s then in running water and peel s and stuff s them . Pol ly drain s the potatoes , and peel s them . none Zak arrange s the vegetables on a platter and covers them . Meanwhile , m ama Mary am packs the plates , glasses , implement s and linen s in a wick er hamper , and pa pa Scott packs the food and drinks in a cooler . The family is ready for their picnic .
Spinn3r
Some guy called me early this morning . Let ' s just say the conversation didn ' t enlighten him the least bit , because he woke me up , and I was gro ggy and annoyed . Set my phone to silent mode the night before , and when he called , it sort of reminded me of my alarm , except more irritating , because I couldn ' t hit the s no o ze button and stop it from vibr ating for minutes , like what I always do to the alarm . Him : j nt gi b n w v bu j f v b q Could n ' t really catch what he was saying Me : Hello ? Damn you , do you know what time it is ? ! I checked the clock Me : It ' s in the morning ! Him : an m ku y wi q l Could n ' t hear what he was mum bling about * Then it suddenly struck me that I had no clue who he was . O ops . Check ed the Call er ID thing , but the number was with held . Oops again . Then I hung up the phone , and went back to sleep . ...

Table 6.1: Sample stories from different corpora. InScript and MCScript consist of scenario descriptions, while Spinn3r comprises naturally generated personal blogs

model	gold regularity	events		participants	
		macro F1	micro F1	macro F1	micro F1
Bare XLNet	X	24.1	25.0	17.8	19.5
w/o dep, coref, specialized	X	$44.2_{\pm 0.57}$	$51.2_{\pm 0.83}$	$37.0_{\pm 1.4}$	$39.0_{\pm 0.72}$
w/o dep, coref	X	$45.7_{\pm 0.63}$	$50.2_{\pm 0.63}$	$37.9_{\pm 1.1}$	$41.9_{\pm 1.0}$
w/o dep	X	$48.2_{\pm 1.4}$	$53.3_{\pm 1.3}$	$40.8_{\pm 1.8}$	$45.4_{\pm 1.7}$
Full	X	$\mathbf{49.6}_{\pm 1.3}$	$\mathbf{55.5}_{\pm 1.6}$	$\mathbf{42.6}_{\pm 1.6}$	$\mathbf{54.0}_{\pm 2.0}$

Table 6.2: Results on MCScript, averaged from five parallel training runs.

6.2 Script Acquisition on MCSCript

6.2.1 Data

MCScript

MCScript is a question answering dataset that focuses on script knowledge. As a background of its questions, it has around 20 stories for each of 200 scenarios. These stories are stylistically similar to those in InScript, as they also consist of relatively simple language and focus on explaining the scenario in detail (see Chapter 2 for more details). We use these background stories as a starting point for script acquisition via the zero-shot parser. In particular, we will train the zero-shot parser on InScript, and perform reference on MCScript to acquire script knowledge.

Annotation

As a question answering dataset, MCScript is not annotated with script events and participants. For evaluation purposes, we need annotated MCScript stories. To this end, we select 20 scenarios. We select these scenarios according to the following guidelines. First, the set of scenarios we select does not overlap with InScript scenarios, as they are supposed to be unseen during the training of the parser. Second, each of these scenarios should not be too similar to any other. To perform the selection, we randomly choose from the set of scenarios, one at a time, and add it to the set of selected scenarios if it does not violate any of the guidelines.

The annotation process is similar to that of InScript. First, we read a number of stories and determine the set of events and participants (originally termed *templates* by Modi et al. (2016)). Second, following the original paper, we annotate the stories with these event and participants, and add more labels to the set if it turns out

Dataset	Tokens	Stories	Events	Irregular Events	Participants	Irregular Participants
validation	28271	154	7868	1537	3868	1831
test	28429	154	8113	1458	3784	1540
val + test	59700	308	15981	2995	7652	3371

Table 6.3: Stats of the annotated part of MCScript

that is necessary. We collapse all irregular event or participant classes into one, that is, either irregular event or irregular participant. To ensure better quality, several stories that are less suitable are excluded from the annotation. Most of the exluded stories suggest some misinterpretation of the scenario. The annotation is performed by two experts. Table 6.3 provides some statistics of the annotated part of MCScript. Overall, the annotations are similar to those of InScript. But given the simplified annotation pipeline and that we have less than 20 stories per scenario as opposed to 100 in InScript, the annotated MCScript should be considered as a different dataset.

10 of these annotated scenarios are selected as the validation set and the other 10 as the test set.

6.2.2 Experiments

We follow the pipeline in Chapter 5 to train a zero-shot parser, but we made minor adjustments. Essntially, we use the entirety of InScript as our training data, instead of eight out of ten scenarios, as we do in Chapter 5. We validate with the MCScript validation set and test on its test set. This change also applies to training of the regular candidate identifier and specialized embeddings, when applicable. The optimization is also rerun with new hyper-parameters.

Results

We apply the same evaluation method as in the previous chapter. XLNet and the same ablated models (without dependency, coreference, or specialized embeddings) are included for comparison purposes. The results are given in Table 6.2. Our parser achieves around 50 point average F1, in contrast to the 70 points when evaluated on InScript. This is well anticipated, as we have much less data available: the test scenarios only have around 20 stories each, as opposed to the 100 stories for each InScript scenario. More importantly, although similar, InScript and MCScript are collected differently and have undergone different annotation processes. The performance drop is especially noticeable for participant parsing, which was unexpected. A closer look indicates that participants in MCScript scenarios follow much more skewed distributions than in InScript: instances from the less frequent half of the participant classes in MCScript take a proportion of only 8%, whereas this quantity is 18% for InScript. As a result, there are more classes that only have a handful of instances than InScript, worsening the already lower-resource setting.

All our model variants still outperform the XLNet baseline by a large margin, with the full model achieving the best performance, at 49.6/55.5 macro / micro F1 for events and 42.6/54.0 macro / micro F1 for participants. All model components contribute to this performance according to the ablation study.

Table 6.4 shows some sample clusters produced by our zero-shot parser for the going jogging scenario. The event cluster mostly collects candidates annotated as return. There are a few excpetions. We notice three candidates annotated as finish run in this cluster. The event finish run refers to the action of stopping running, as many people would perform other activities before return home, such as stretching, drinking water, etc. However, if we look closer at these three instances, we found it reasonable to put them in the cluster of return: these candidates are *head home*, *drive home* and *get home*. Furthermore, in these three stories, no activities are described between running and returning home. That is, in these stories, finishing running and returning home are described with one verb only. For these stories, an event set that collapses those two events is more suitable. Once again, this is a *granularity* issue. The same applies to the second candidate labeled jog(...i *made* it back home), which describes the latter half of jogging until arriving home in one verb. The participant cluster is a relatively clean correspondence of the participant company produced by the annotation. The overall quality is good, given how little data we have and that no scenario-specific annotation was available. With the clustering results, we could also compile temporal script graphs of MCScript scenarios. The TSGs are in good shape, in general, yet, with fewer stories, they could be noisy and indecisive. See Figure 6.1 for an example.

6.3 Wider Coverage Story Generation

In Section 6.2 we apply our zero-shot parser to MCScript and perform an intrinsic evaluation, by checking Hungarian F1 and analyzing the output clusters. The purpose of developing the zero-shot parser is to automate the annotation process.

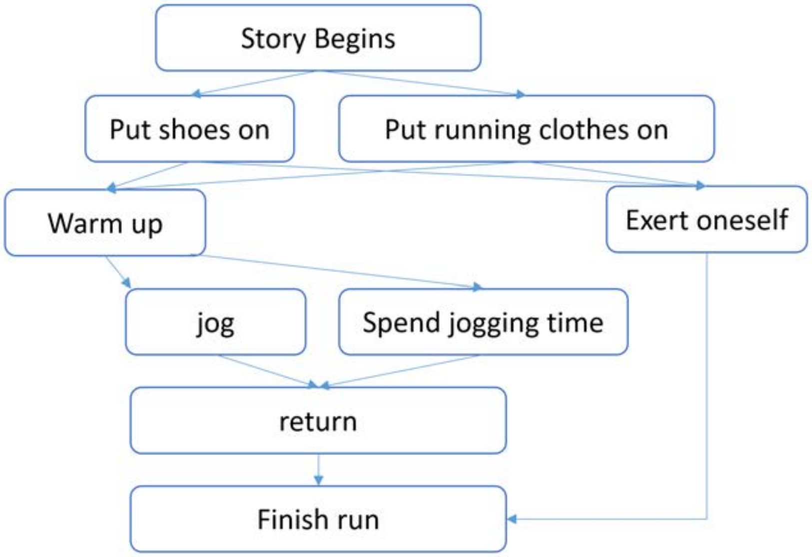


Figure 6.1: A fragment of the temporal script graph for going jogging inferred from our parsing result. Cluster indices are replaced with event names according to the annotation on the test set and the alignment produced by the Hungarian algorithm. exert oneself sees a granularity issues, as people might stretch in different phases of jogging for slightly different purposes.

ground truth	text
go to jogging place	...I stood up next to the girl I had [come] to the park with and said , “ Ok...
return	...get some water at the water fountain there . Then I [head] back home , and take a nice cool shower...
return	...have everything I need , because I hate the idea of [coming] back to the house before I ’ m finished...
return	...That was when I knew it was time to [go] back home . I jo gged back to...
jog	....I then ran the whole 3 mile loop . I [crossed] the street into my neighborhood again and ran through my neighborhood...
return	...When we finished jogging , I [returned] home and gave my dog a treat , then...
return	...out on their morning run . I finish my jog and [head] home where I take off my shoes , and shower...

return	...with every step . After about twenty minutes , I [turned] around to head back home ....
return	.... After about twenty minutes , I turned around to [head] back home ...
finish run	...if I jog only 3 times per week . Then I [drive] home and shower...
go to jogging place	...fill it with water . I get into my car and [drive] to the park...
finish run	...After an hour of jogging I go back to my car and [head] home for breakfast...
finish run	before finishing my jog for the day . When I [get] home , I make sure to take a shower...
return	...could sustain . none About 15 minutes into my run I [turned] around and headed home...
return	.... none About 15 minutes into my run I turned around and [headed] home...
jog	...into my run I turned around and headed home . I [made] it back home in about 15 minutes...
company	...I stood up next to the [girl] I had come to the park with and said...
company	...“ Ok , I ’ m ready when [you] are . ” ...
company	...I sped up just a little bit to catch up with the [girl] . Down the path we jo gged ...
company	...but I could n ’ t help but wonder if this [girl] ever going to run out of breath ...
company	...we started out . I was amazed at how effort less [she] was able to glide through the air...
company	...with only the slightest hint of sweat gli sten ing off [her] brow ...
company	.... Finally , after two hours of keeping up with [her] rhythmic pace I could n ’ t...
company	... in the glare of the midday light I see [her] face with a crooked little smirk on it...
irregular	...midday light I see her face with a crooked little smirk on [it] as she says...
company	...I see her face with a crooked little smirk on it as [she] says , “ All done with our warm up jog...

Table 6.4: Example output clusters. Top: event; bottom: participant. The candidates are highlighted with brackets. The table presents **all** instances from these clusters.

Therefore, in this section, we evaluate its performance as an *annotator* by training models for a downstream task based on the *annotation* it provides. The task is story generation, which we investigate extensively in Chapter 3. More specifically, we re-run the pipeline in section 3.3 on MCScript, using the output of our zero-shot parser as the source of script knowledge. We shall see that automatically acquired script knowledge is already sufficient to accommodate the training of a story generation model, whose performance approaches that of a model trained on human annotated corpus.

6.3.1 Data

To train the story generation model presented in section 3.3, we need, for each scenario of interest, stories annotated with event types. What we assume to have access to are MCScript stories that do not have annotations. To acquire these event types, we run the zero-shot parser as trained in the previous section to yield a partition of the event-evoking tokens, putting together those considered to evoke the same event. Now, equivalently, we have some *annotated* data to train the model with.

For the sake of clarity, we borrow the event cluster names from the annotations of the MCScript test set to substitute the event indices, which are meaningless integers. Recall that during the evaluation of Hungarian F1, we also acquired an injection from the partition to the set of events. Thus, for an event where the injection is defined, we replace the cluster index with the name of the event to which its cluster maps. As a result, we get InScript-style data on the 10 test scenarios of MCScript, whereas the annotations are purely automatic. We process the data with the same segmentation process as is described in Figure 6.5, to establish the correspondence between the instantiations of events and the segments of the surface text. Figure 6.5 provides some example segments.

6.3.2 Experiments

We now have our data from the MCScript test set appropriately formulated to train the model in Section 3.3. Note that the test set of MCScript has, on average, 15 stories per scenario, whereas InScript has around 100. The shortage of data has two consequences for our experiment. First, the model cannot be sufficiently trained. In fact, it would turn out that the MCScript test set does not offer enough data to train our story generation model, as a model trained with this amount of data can barely generate sensible texts. Second, the validation set is also quite small, with

event	text segment
drive	I was driving home from work when I noticed I was low on fuel .
run out of gas	I was worried about running out of gas , so I stopped at the next available station .
drive to pump	I pulled my car up to the pump , mindful of the side of the car my gas tank was on .
get out of car	I turned off my car and got out .
pay	I gave the cashier a twenty dollar bill and told her I wanted twenty dollars of gas on the pump I was at .
pay	She entered it in and I walked back to my car .
open tank lid	I got to my car and opened the fuel tank .
pay	I took the hose off of the pump and chose my fuel ,
close tank lid	then I put the hose into my car .
turn pump off	As it got closer to the twenty dollars , it slowed down .
turn pump off	I stopped the pump at twenty dollars .

Table 6.5: The segmentation of a story on fueling a car according to the rules in Figure 3.9.

which the training is not stable.

To address this issue, we decide to expand the training data to include the entirety of InScript. Furthermore, we include 50% of the MCScript test set in the training data; the other half is kept as validation data. No test set is necessary, as our evaluation is performed by assessing the quality of generated stories with human annotators. Note that now InScript makes up the vast majority of training data (around 90% in terms of tokens), while the entire validation set comprises stories from MCScript. Intuitively, the model is supposed to develop an *InScript-style* generation behavior; however, it will focus on writing stories in MCScript scenarios, as no InScript story is included in the validation set.

Note that the zero-shot setting is not compromised. It is true that a story generation model trained with our settings benefits from the annotations in InScript, but we are still generating stories on scenarios that had no manual annotation. In fact, it is assumed that the entire InScript is accessible, as the zero-shot parser per se is trained on InScript.

The optimization is identical to that in Chapter 3, yet we rerun the random hyper-parameter search to select a new set of hyper-parameters.

No script knowledge, cherry-picked
i took a garbage in my room and dumped the garbage inside . i open the lid of the garbage can and take out the bag . it was a full of misplaced wrapper s on the garbage . i went to my end of the dump ster , where the garbage man will be able to take .
Scenario: taking out garbage. ***Agenda***: N / A
Zero-shot script knowledge
my little cousin asked me to read out a story . he looked through the shelves for a minute and pulled out a thin book full of colorful pictures . we walked over to the couch and sat down . i flipped past the title page and turned the book to him to show him the pictures . she like s to read it with me .
Scenario: reading a story to a child. ***Agenda***: *evoking → let child choose story → turn page → read → child reads along → put book away*
Human annotated script knowledge
Before heading out for a jog, it is important to put on the proper shoes and to warm up the muscles. Once the muscles are warmed up, it is time to start jogging at a slow and steady pace. After a few minutes, it is okay to take a break and walk for a minute or two to catch ones breath. After the break, it is time to start jogging again and to continue at a steady pace until the end of the workout.
Scenario: going jogging. ***Agenda***: *story begins→put shoes on →warm up→jog →take break →story ends*

Table 6.6: Stories generated by the model in Section 3.3, trained with data that has different accessibilities to script knowledge.

Source	Coverage	Syntax	Global Coh.	Local Coh.	Relevance	Informativeness
No S. Knowledge	3%	.25	.07	.24	.07	.07
Zero-shot	50%	**.56**	**.57**	**.54**	**.63**	**.59**
Annotated	**63%**	.43	.49	.42	.59	.51

All differences are statistically significant according to independent T-test at $\alpha = 0.05$.

Table 6.7: Results from human evaluation. All metrics are ranged $[0, 1]$.

6.3.3 Evaluation

We evaluate the stories generated in the same way as in Chapter 3, that is, by crowd-sourcing. Several systems are included in the evaluation.

No-Script-Knowledge. In this version of the model, we do not distinguish between events. In practice, all event types are replaced with the respective scenario names, while the model architecture remains intact. In consequence, the correspondence between script knowledge and surface text is neutralized; therefore, the model has no access to even automatically annotated script knowledge.

Zero-shot. This is the model whose training supervision comes from our zero-shot parser.

Annotated. For this one we execute the same pipeline as Zero-shot Script Knowledge, but the training supervision is not from our zero-shot parser, but from the manual annotations in MCScript's test set.

Table 6.7 gives some examples of generations from these systems. The results of the human evaluation are given in Table 6.7.

Without script knowledge, the quality of generation is pretty poor. The output of No Script Knowledge is barely grammatically correct. The fact that it still gets scenario information allows it to be on topic more often than random. As we include InScript scenarios in the training data, which are even overwhelming in size (approx. 5 times), the model is not always good at staying on the scenario it is supposed to describe. Actually, a considerable proportion of the generated texts are not completely readable. The overall poor quality is reflected in a very low score in all aspects of evaluation.

Based on the automatically annotated data, Zero-shot Script Knowledge is able to generate coherent stories on the given scenario. We see a substantial improvement over the baseline in terms of human evaluation. Its performance is close to that of Human Annotated Script Knowledge. Yet, a closer inspection shows that not all credit goes to the quality of generated text: the zero-shot learning is performed on the scenarios that have less than 20 stories each. Therefore, some less frequent but valid events might not be evoked regularly enough to result in a cluster. The situation is different in human annotations, as an annotator also recognizes events based on his commonsense knowledge. As a result, we obtain smaller sets of events for Zero-shot Script Knowledge, which makes the generation task in general less complicated. All in all, script acquisition from our zero-shot parser proves sufficient in accommodating the task of story generation.

6.4 Script Acquisition on Spinn3r

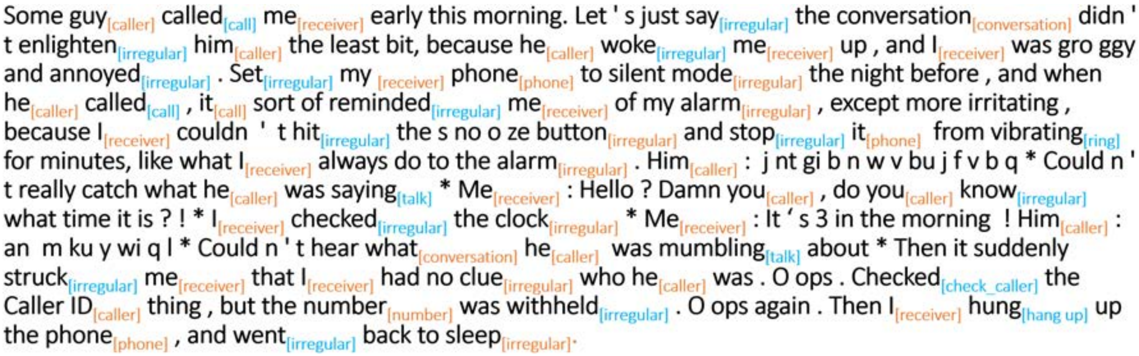
In the previous section, we performed script acquisition on MCScript, based on our zero-shot parser; we show that script knowledge acquired on MCScript via our zero-shot parser is sufficient to support the downstream task of story generation. However, the stories in MCScript are crowd-sourced, which still incurs a considerable amount of cost. In this section, we explore the potentiality of script acquisition on freely accessible, naturally generated texts with the methods we have developed.

The stories in MCScript differ from those generated naturally mainly from two perspectives. First, these texts are simple in language, since crowdsource workers were instructed to tell the story *as you would tell a child*; second, all MCScript stories are devoted to describing the scenario of interest, while this is usually not true for naturally generated texts, because script knowledge is assumed to be common knowledge among the authors and the readers. As it would turn out, script acquisition from naturally generated texts is much more complicated than from MCScript; our parser seem to exhibit sensible behavior but is insufficient to substitute human annotators.

6.4.1 Data

Our experiments are conducted on a subset of *Spinn3r* (see Burton et al., 2009), which is a collection of around 44 million personal blogs. Gordon and Swanson (2009) identified 1.5 million from the corpus as *personal stories*. These are supposed to be narratives on daily activities; these narratives are supposed to, to some extent, encode script knowledge (see Figure 6.1 for an example story from Spinn3r). From the perspective of script acquisition, naturally generated texts present a very different picture than InScript. As script knowledge is a category of commonsense knowledge, one would rarely describe it in detail, but would most probably assume that the readers are already aware of it and focus on the informative parts of the story. These incomplete descriptions drastically improve the difficulty of acquiring procedural knowledge.

As our zero-shot parser works on multiple descriptions of specific scenarios of interest,



Some guy[caller] called[call] me[receiver] early this morning. Let ' s just say[irregular] the conversation[conversation] didn ' t enlighten[irregular] him[caller] the least bit, because he[caller] woke[irregular] me[receiver] up , and I[receiver] was gro ggy and annoyed[irregular] . Set[irregular] my [receiver] phone[phone] to silent mode[irregular] the night before , and when he[caller] called[call] , it[call] sort of reminded[irregular] me[receiver] of my alarm[irregular] , except more irritating , because I[receiver] couldn ' t hit[irregular] the s no o ze button[irregular] and stop[irregular] it[phone] from vibrating[ring] for minutes, like what I[receiver] always do to the alarm[irregular] . Him[caller] : j nt gi b n w v bu j f v b q * Could n ' t really catch what he[caller] was saying[talk] * Me[receiver] : Hello ? Damn you[caller] , do you[caller] know[irregular] what time it is ? ! * I[receiver] checked[irregular] the clock[irregular] * Me[receiver] : It ' s 3 in the morning ! Him[caller] : an m ku y wi q l * Could n ' t hear what[conversation] he[caller] was mumbling[talk] about * Then it suddenly struck[irregular] me[receiver] that I[receiver] had no clue[irregular] who he[caller] was . O ops . Checked[check_caller] the Caller ID[caller] thing , but the number[number] was withheld[irregular] . O ops again . Then I[receiver] hung[hang up] up the phone[phone] , and went[irregular] back to sleep[irregular].

Figure 6.2: A Spinn3r story on answering the phone annotated with script events and participants. As is opposed to InScript stories, this story is about a personal experience, instead of the general knowledge about what it means to answer a phone.

the first step is to identify scenarios of interest from Spinn3r's personal stories. There are existing resources that fit our need. Wanzare et al. (2019) approach scenario detection on Spinn3r stories. For evaluation purposes, they annotated around 500 stories with scenario labels. Wanzare et al. (2019) firstly segment the stories with topic modeling so that each segment is devoted to a single scenario. The segments are then annotated with scenario labels. Therefore, the starting point of our approach is established, that is, texts on specific scenarios. Yet, these texts do not have event nor participant annotations. We take about 10% of these texts and manually annotate them with event and participant types, to evaluate our script parser. These texts are about two scenarios that are not included in InScript: answering the phone and working in the garden. Figure 6.2 gives a sample story from Spinn3r with its annotations.

6.4.2 Results

We execute exactly the same pipeline as we did on MCScript. That is, we train our zero-shot parser on InScript, to perform inference on Spinn3r. We also evaluate its performance with the Hugarian F1 we defined in the previous chapter. The quantitative results are given in Table 6.8.

Unsurprisingly, we see a considerable performance drop compared to MCScript. These datasets are close in size. The most important reason for the performance difference is that Spinn3r texts do not encode script knowledge in a structured manner, like MCScript. For example, unlike elaborating how one would typically

model	gold regularity	events		participants	
		macro F1	micro F1	macro F1	micro F1
XLNet	X	29.0	32.2	26.4	29.1
Zero-shot	X	31.6	35.7	27.2	35.9

Table 6.8: Results on Spinn3r. Both models inference based on the regularity predictions by our regular candidate identifier, so XLNet also benefits from it.

answer a phone, the story in Example 6.1 is about a fun experience that is relevant to answering a phone. Therefore, the text rarely strictly mentions knowledge about answering the phone, but assumes it. In contrast, InScript stories are completely devoted to describing the scenario, with a roughly constant flow of information.

We still observe moderate performance improvement over XLNet. That means some knowledge on script parsing is indeed learned from InScript and transferred to Spinn3r.

Table 6.4 gives a coupld of cherry-picked clusters produced by our parser. In general, the parser exhibits promising performance on some of the most obvious clusters, like the ones shown in Table 6.4. However, it really struggles to come up with reasonable clusters for the less obvious ones. We believe that with script knowledge more implicit in text, the model needs much more data from Spinn3r than from MCScript to deliver a similar performance.

ground truth	text
receive call	...day my home phone rang and my husband s cell phone number [came] up on the caller I . D . But when...
call	...She also had [called] my husband s cell phone before she called me and he...
call	... as our home number . She has [called] twice since then , the second time she called I receive...
call	...phone rings last night and my husband s cell phone [shows] up , funny thing is my husband was sitting right...
phone	...our date was winding down , Justin ' s cell [phone] rang . The first thought was ,...
phone	...The other day my home phone rang and my husband s [cell] phone number came up on the caller I ...
phone	...The other day my home phone rang and my husband s cell [phone] number came up on the caller I ...
phone	...agency) . She also had called my husband s [cell] phone before she called me and he said it came up...
phone	...before she called me and he said it came up on his [caller] I . D . as our home number ....
call	...twice since then , the second time she called I answered [it] thinking it was my husband but it was her so I...
caller	...time she called I answered it thinking it was my husband but [it] was her so I hung up . Then the phone...
phone	...funny thing is my husband was sitting right beside me and his [cell] phone was off , so we let the voice mail...
phone	...thing is my husband was sitting right beside me and his cell [phone] was off , so we let the voice mail pick...
call	... So back on the 20 th of May I got a [call] from a scam ming company trying to give me a ...

Table 6.9: Example output clusters from answering the phone, acquired via zero-shot script parsing on Spinn3r. Top: event; bottom: participant. The candidates are highlighted with brackets. Note that there is an event call, as well as a participant call. The table presents **all** instances from these clusters. These clusters are casually cherry-picked.

Chapter 7

Outlook

In this chapter, we discuss the future of explicit script acquisition.

The most important question is, what role could large pretrained language models play in the acquisition of script knowledge?

Pretrained language models have continuously exhibited impressive performance in various NLP tasks. In particular, existing research efforts show that pretrained language models encode knowledge about our world and that the knowledge could be extracted (see, e.g. AlKhamissi et al., 2022; Petroni et al., 2019). We have good reasons to anticipate that script knowledge, which is ubiquitous in language practices, is also, to some extent, encoded in these models. That opens up two possibilities. First, we could use them as the source of script knowledge to perform script acquisition and extract key elements of script knowledge, such as events and participants. Second, we could try to design script-knowledge-aware NLP applications based on pretrained language models. Such applications will not access script knowledge in any explicit format, but behave so that they do not violate script knowledge. Potentially, this approach could avoid the technical difficulties induced by explicit script acquisition. For example, granularity will no longer be a major concern as we will not have to dictate what the sets of events and participants are.

In this chapter, we take a step towards answering this question: how much script knowledge is encoded in pretrained language models? In particular, is it abundant enough to render explicit script acquisition unnecessary?

We start this chapter with an introduction to large pretrained language models (Section 7.1). Afterwards, we inspect a prominent example of pretrained language models, GPT-3, from two perspectives. First, we query GPT-3 to acquire key elements of script knowledge, including events, participants, etc. We also attempt to let GPT-3 perform script parsing (Section 7.2). Second, we evaluate it as a story generation system, to compare with what we developed in Chapter 6 (Section 7.3). As we shall see, our inspections suggest that GPT-3 could deliver impressive performances. At the current state of research, pretrained language models seem to offer better performance than explicit script acquisition. These observations are consistent with recent works of the community.

7.1 Pretrained Language Models

Large language models play a prominent role in modern NLP. Access to unprecedented amounts of training data and computational resources grants high-quality general-purpose representations. A product of our era, pretrained language models deliver strong performances on various tasks, whereas requiring only moderate development expenses. These models encode both linguistic knowledge and common sense knowledge (see, e.g., Bouraoui et al., 2020). From the perspective of this thesis, the most interesting question is how much script knowledge is encoded in these models. In this section, we briefly overview some models from the transformer family and how pretrained language models could be used in zero-shot style learning.

7.1.1 The Transformers

The term *transformer* originates from Vaswani et al. (2017). The authors proposed a novel neural network architecture, which they term *transformer*, to construct a sequence-to-sequence translation model. The model, which claims *attention is all you need*, unconventionally stripped most of the reccurrence features in sequence models like LSTM(see Hochreiter and Schmidhuber, 1997; Sak et al., 2014). The model improved both the state of the art of English-to-German and that of English-to-French by a considerable margin.
A transformer is based almost exclusively on attention mechanisms. A transformer encoder is a chunk of neural network consisting of multiple transformer layers. Each transformer layer takes a sequence of vectors as input and converts them into a sequence of essentially the same length. At the first layer, the input comprises the embeddings of each word piece. These layers feature the following technical designs.

Multihead attention. Attention is the central mechanism of a transformer model. Each layer contains multiple attention mechanisms that share the same design. In each of these 'heads', each input is associated with 3 vectors: a d_k dimensional key K, a quiry R of the same size, and a value V. To evaluate attention from one position to the other, we use the quiry vector of the former to quiry the key vector of the latter:

$$Att(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

The attention quantities of all attention heads are concatenated, which, after going through a residual connection and layer normalization, is fed to a fully connected layer. The outcome is then fed to the next transformer layer.

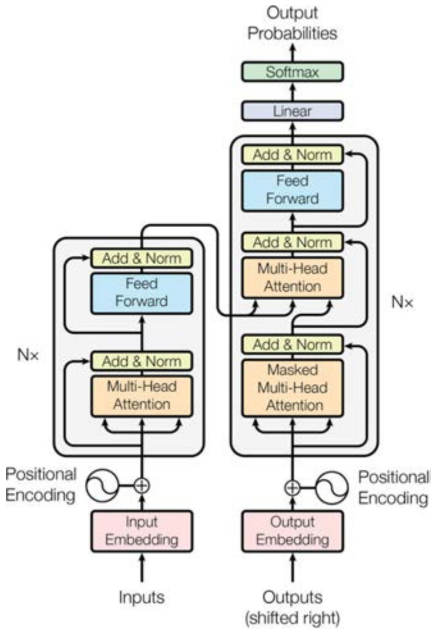


Figure 7.1: Depiction of the original transformer model from Vaswani et al. (2017)

Positional Encoding Multi-head attention as is defined above is ignorant of word order. To this end, a positional encoding vector is added to each input vector to leave a trace of its position in the input sequence. The positional encoding for the token at position p is defined as

$$\mathcal{P}(p, 2i) = sin(p/10000^{2\pi/d})$$

$$\mathcal{P}(p, 2i+1) = cos(p/10000^{2\pi/d})$$

In some later versions of transformers, positional encoding becomes trainable.

The community has seen a considerable number of variants of the original transformers. For example, BERT (Kenton and Toutanova, 2019) trains on a pair of input sequence and use the masked language model objective to predict each token based on both its left and right context, to acquire general purpose representations; XLNet (Yang et al., 2019) introduces sequence level recurrence to remove the limitation of the input length; RoBERTa (Liu et al., 2020) further refines the training process.

7.1.2 Transfer Learning from Large Language Models

Large language models see an unprecedented concentration of resources in NLP. In 2019, BERT, which has 340M parameters, was trained with a corpus of 3.3 billion

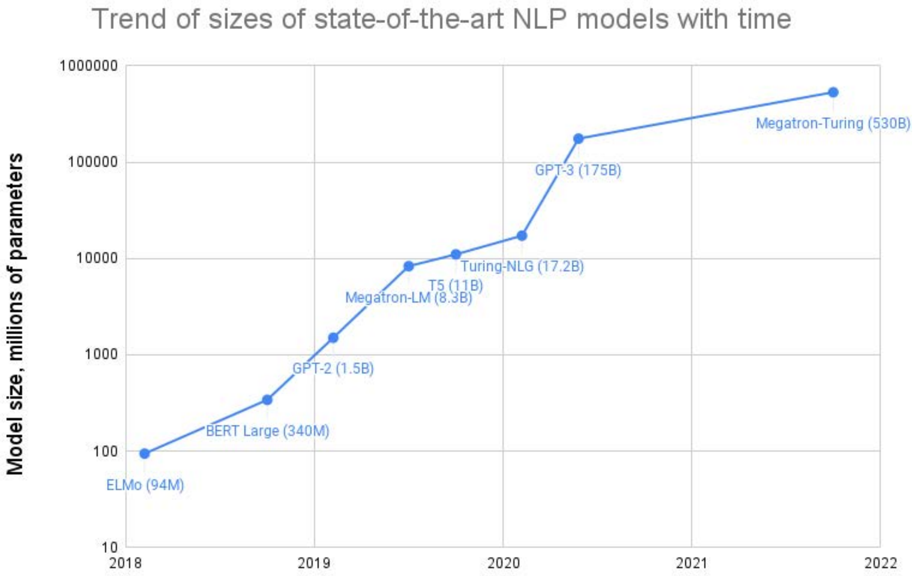


Figure 7.2: The scales of pretrained language models. Taken from `https://medium.com/pytorch/training-a-1-trillion-parameter-model-with-pytorch-fully-sharded-data-parallel-on-aws-3ac13aa96cff`

words (Kenton and Toutanova, 2019); GPT-2 (Radford et al., 2019) was trained with 40GB text and has 1542M parameters. In 2020, GPT-3 (Brown et al., 2020) raised the numbers to around 1 trillion works and 175 billion parameters. The training cost of GPT-3 is estimated to be around 12 million US dollars[1]. See also Figure 7.2.

Since these language models have access to such an incredible amount of data and computational resources, it is reasonable to anticipate that they encode some general knowledge of linguistics and of our world. In fact, transferring this knowledge has been a major approach to improving the state of the art in various NLP tasks (see, e.g., Kenton and Toutanova, 2019; Brown et al., 2020), without requiring much training supervision. The most straightforward method, as proposed in Kenton and Toutanova (2019), is to use these models as a feature extractor, to train classifiers for the task of interest. Another approach is *fine-tuning*, as proposed by Howard and Ruder (2018), where a small learning rate is applied to reinstate training to adapt to a specific task.

---

[1] Elliot Turner, at `https://twitter.com/eturner303/status/1266264358771757057`

Whereas fine-tuning a pretrained model still requires a certain amount of annotated training data, *prompting* allows researcher to further cut the requirement on annotated data; it is even feasible to learn some tasks without annotated data at all. By promting we feed a language model a (usually carefully designed) piece of text, aka the prompt, and ask it to continue the text. The continuation, together with the prompt, is supposed to receive a high probability from the language model. Which means, the text should be *similar* to the training data, module the model's inductive bias. There are various aspects when it comes to being similar. The generated text of a language model could be similar to its training data by being grammatical, by not violating kommonsense knowledge, etc. Yet surprisingly, the latest language models' generations exhibit such coherence that it seems that the language model begins to understand *task descriptions*. That means we could directly query these models for answers. We could either directly consider the language model a solution to our problem of interest, or at least generate quality auxillary training data from it (see, e.g., Radford et al., 2019; Schick and Schütze, 2021). Here are some examples[2].

Prompt: *POS tagging.*
Input: The FBI searched Donald Trump's Mar-a-Lago resort in Florida after it obtained evidence there was probably an effort to conceal classified documents in defiance of a grand jury subpoena and despite his lawyers suggesting otherwise, the justice department said in a court filing.

\>> *Output: The/DT FBI/NNP searched/VBD Donald/NNP Trump's/NNP Mar-a-Lago/NNP resort/NN in/IN Florida/NNP after/IN it/PRP obtained/VBD evidence/NN there/EX was/VBD probably/RB an/DT effort/NN to/TO conceal/VB classified/JJ documents/NNS in/IN defiance/NN of/IN a/DT grand/JJ jury/NN subpoena/NN and/CC despite/IN his/PRP$ lawyers/NNS suggesting/VBG otherwise/RB ,/, the/DT justice/NNP department/NN said/VBD in/IN a/DT court/NN filing/NN ./.*

---

[2] These examples are generated with openAI's demo, under the default settings: `https://beta.openai.com/playground`. This also applies to the stories we evaluate in Section 7.3. Due to availability issues, some examples in Sections 7.2 and 7.3 are generated with ChatGPT in `https://chat.openai.com/`.

Prompt: *NER tagging.*
Input: The FBI searched Donald Trump's Mar-a-Lago resort in Florida after it obtained evidence there was probably an effort to conceal classified documents in defiance of a grand jury subpoena and despite his lawyers suggesting otherwise, the justice department said in a court filing.

\>> *Output: <B-NER>Donald Trump</B-NER>'s Mar-a-Lago resort in <B-LOC> Florida</B-LOC> after it obtained evidence there was probably an effort to conceal classified documents in defiance of a grand jury subpoena and despite his lawyers suggesting otherwise, the <B-ORG> justice department</B-ORG> said in a court filing.*

Prompt: *Noun phrase chunking.*
Input: The FBI searched Donald Trump's Mar-a-Lago resort in Florida after it obtained evidence there was probably an effort to conceal classified documents in defiance of a grand jury subpoena and despite his lawyers suggesting otherwise, the justice department said in a court filing.

\>> *Nounphrase chunking:*
The FBI
Donald Trump's Mar-a-Lago resort
Florida
evidence an effort to conceal classified documents
defiance
a grand jury subpoena
his lawyers
the justice department
a court filing.

Prompt: *Barack Obama was born in*

\>> *Hawaii in 1961, to a white mother and black father.*

We could see that GPT-3 responds pretty impressively to several foundational NLP tasks. It also exhibits some knowledge about the world. In the remainder of this chapter, we inspect its capacity as a source of script knowledge.

7.2 GPT-3 as a Source of Script Knowledge

Scenario Description

First, we examine whether GPT-3 could generate accurate descriptions of script scenarios. That is, if it could generate text that provides a correct description of the activity, like the texts we see in WikiHow (`https://www.wikihow.com/`). Being able to generate such texts would preliminarily show that the model encodes script knowledge. Since script knowledge is ubiquitous in naturally generated texts, we anticipate that correct descriptions of script scenarios are frequent on the Internet. That means they are also frequent in the training data of GPT-3, as a large proportion of the training data is from common crawl. Such texts are easy to learn by large language models, so GPT-3 should be able to generate decent scenario descriptions.

(a) Prompt: *Describe how to change a light bulb.*

>> *1. Turn off the power: Before changing a light bulb, make sure to turn off the power to the light fixture. You can do this by flipping the switch off or by turning off the circuit breaker that controls the light.*

2.Let the bulb cool down: If the bulb has been on recently, let it cool down for a few minutes before attempting to remove it. Hot bulbs can break easily and can also burn your fingers.

3.Remove the old bulb: If the bulb is in a lamp or a ceiling fixture, gently twist it counterclockwise until it comes loose from the socket. If the bulb is in a recessed or track lighting fixture, you may need to push up on the bulb and turn it counterclockwise to remove it.

4.Choose the right replacement bulb: Check the wattage and size of the old bulb to ensure you purchase the correct replacement bulb.

5.Insert the new bulb: Holding the new bulb by the base, gently insert it into the socket and turn it clockwise until it feels snug.

6.Turn on the power: Once the new bulb is in place, turn the power back on to the light fixture by flipping the switch or turning on the circuit breaker.

7.Test the new bulb: Turn on the light to make sure the new bulb is working properly.

(b) Prompt: *Describe how to fix a flat tire.*

\>> *Fixing a flat tire is a straightforward process that requires a few tools and some basic knowledge. Here's a step-by-step guide on how to fix a flat tire:*

1.Find a safe spot: If you realize that you have a flat tire while driving, try to find a safe spot to pull over. Avoid busy roads, sharp turns, and hills.

2.Turn on the hazard lights: Once you have found a safe spot, turn on the hazard lights to signal to other drivers that your vehicle is not moving.

3.Gather the tools: You will need a few tools to fix a flat tire, including a spare tire, a lug wrench, and a jack. These tools are typically located in the trunk of your car.

4.Loosen the lug nuts: Using the lug wrench, loosen the lug nuts on the flat tire. Do not remove them yet.

5.Lift the vehicle: Use the jack to lift the vehicle off the ground. Make sure that the jack is placed on a solid surface and that it is supporting the weight of the car.

6.Remove the flat tire: Once the vehicle is lifted, remove the lug nuts and take the flat tire off.

7.Install the spare tire: Place the spare tire onto the wheel hub and screw on the lug nuts by hand.

8.Tighten the lug nuts: Using the lug wrench, tighten the lug nuts in a star pattern until they are snug.

9.Lower the vehicle: Slowly lower the vehicle back to the ground using the jack.

10.Tighten the lug nuts again: Using the lug wrench, tighten the lug nuts in the same star pattern until they are fully tightened.

11.Check the tire pressure: Once the tire is fixed, use a tire gauge to check the pressure of the spare tire. The recommended pressure can be found in your car manual or on the tire itself.

> *12.Put away the tools: After the tire is fixed, put away all the tools and the flat tire in the trunk of your car.*
>
> *By following these steps, you should be able to fix a flat tire quickly and safely. However, if you are unsure about any of these steps or if you do not feel comfortable changing a tire, it is best to contact a professional.*

These are quality responses. Now we take it one step further by asking for script descriptions under more specific circumstances, to see if the generated descriptions could fit these specificities. For the sake of simplicity, we query for shorter descriptions.

(a) Prompt: *Describe how to take a taxi. (less than 150 words)*

>> *To take a taxi, first, find a taxi stand or hail a taxi on the street. Make sure the taxi is available and has a working meter. When you get in, tell the driver your destination and confirm that they know where it is. Fasten your seatbelt and sit back as the driver takes you to your destination. When you arrive, check the meter to ensure you have been charged the correct fare. If you are paying with cash, hand the driver the fare and any tip. If paying with a credit or debit card, wait for the driver to give you the card machine to complete the transaction. Lastly, don't forget to take any belongings with you before exiting the taxi.*

(b) Prompt: *Describe how to take a taxi with a cell phone. (less than 150 words)*

>> *To take a taxi with a cell phone, first, download a ride-hailing app such as Uber, Lyft or Grab. Create an account and link your preferred payment method. Open the app and set your pickup location and destination. Confirm the ride details and wait for a driver to accept your request. Once a driver accepts, you will see their name, photo, and car details in the app. Track the driver's location and estimated time of arrival. When the driver arrives, confirm their identity and get in the car. The driver will take you to your destination and the fare will be charged to your linked payment method. Don't forget to rate the driver at the end of the ride.*

(c) Prompt: *Describe how to call a taxi for 8 people. (less than 150 words)*

\>> *To call a taxi for 8 people, you can either use a ride-hailing app that offers larger vehicles or call a taxi company that provides minivans or SUVs. When calling a taxi company, inform them that you need a vehicle that can accommodate 8 passengers. Provide the pickup location, destination, and desired pickup time. Confirm the cost of the ride and any additional fees. Make sure to get the driver's contact information, including their name and phone number. On the day of the ride, be ready at the pickup location at the agreed time and wait for the driver to arrive. Once the driver arrives, confirm their identity, and ensure that everyone can comfortably fit in the vehicle. Lastly, make sure to buckle up and enjoy the ride.*

(d) Prompt: *Describe how to take a taxi when you have two less than four year old kids with you. (less than 150 words)*

\>> *When taking a taxi with two children under the age of four, it's important to ensure their safety and comfort. You may need to bring your own car seats or booster seats for the children, depending on the local regulations and the taxi company's policies. Call ahead to the taxi company to confirm their policies and ensure that they have the appropriate vehicle to accommodate your family. If using a ride-hailing app, look for options that provide child car seats or booster seats. When you get in the taxi, buckle up the children securely in their seats before securing your own seatbelt. If the children become restless or upset during the ride, try to soothe them and distract them with toys or snacks. Lastly, make sure to double-check the taxi for any items you may have brought with you before exiting.*

(e) Prompt: *Describe how to call a taxi to send a super drunk buddy home. (less than 150 words)*

\>> *To take a taxi with a cell phone, first, download a ride-hailing app such as Uber, Lyft or Grab. Create an account and link your preferred payment method. Open the app and set your pickup location and destination. Confirm the ride details and wait for a driver to accept your request. Once a driver accepts, you will see their name, photo, and car details in the app. Track the driver's location and estimated time of arrival. When the driver arrives, confirm their identity and get in the car. The driver will take you to your destination and the fare will be charged to your linked payment method. Don't forget to rate the driver at the end of the ride.*

(f) Prompt: *Describe how to take a taxi in a country whose language you do not understand. (less than 150 words)*

\>> *Taking a taxi in a country whose language you do not understand can be challenging, but it is possible. Here are some tips:*

1.Prepare the address: Have the address of your destination written down or saved on your phone in the local language. You can also show the address to the taxi driver.

2.Use translation tools: Use translation apps or devices to communicate with the taxi driver. You can use apps like Google Translate or bring a pocket translator with you.

3.Use gestures: Use hand gestures or body language to communicate with the taxi driver. Simple gestures like pointing or nodding can convey a lot of information.

4.Carry a map: Carry a map of the city with you. This can help you orient yourself and give you a general sense of where you are going.

5.Be patient: Remain calm and patient, and try to communicate as clearly as possible. Most taxi drivers are familiar with tourists who don't speak the language, and they will likely be accommodating and helpful.

Overall, taking a taxi in a foreign country may require some extra effort and patience, but with the right preparation and communication, it can be a safe and convenient way to get around.

The model adapts quite well to the different situations in which we need a taxi. Impressively, it generates appropriate, specific descriptions under these situations ((a)-(f)). The texts specify how to handle children, a drunken buddy, and in a foreign country. Now, we push even further, to give it some bizzare situations ((g)-(i)).

(g) Prompt: *Describe how to take a taxi if you are in the middle of no where and it is midnight and your phone is dead. (less than 150 words)*

\>> *If you are in the middle of nowhere and it's midnight with a dead phone, taking a taxi may seem challenging, but it's still possible. Firstly, try to locate the nearest main road or highway and walk towards it. Once you reach the road, look out for passing cars, and try to flag down a taxi or a passing vehicle. If there's no traffic on the road, wait for a while and keep an eye out for any approaching cars.*

If you're lucky, you might find a taxi passing by, and you can wave it down to stop. Once you get into the taxi, inform the driver of your destination, and if possible, negotiate the fare in advance. If you don't have cash, explain your situation to the driver and request him/her to stop at an ATM along the way.

Remember, it's essential to stay safe and be cautious when hailing a taxi from the roadside, especially in the middle of nowhere. Always trust your instincts and take necessary precautions to avoid any potential danger.

(h) Prompt: *Describe how to take a taxi from Amsterdam to Paris, with two pandas and a racoon. (less than 150 words)*

\>> *I'm sorry, but it is not possible to take a taxi from Amsterdam to Paris with two pandas and a raccoon. Taxi services have certain limitations and restrictions, and transporting wild animals is not allowed for safety reasons. Moreover, the distance between Amsterdam and Paris is too long, and it is not feasible to travel by taxi with animals. It is best to explore other options such as taking a flight, driving or using public transportation. If you need to transport animals, it is advisable to contact a specialized pet transportation service that can ensure the safety and well-being of the animals during the journey.*

(i) Prompt: *Describe how to take a taxi when you have no money. (less than 150 words)*

\>> *If you don't have any money to take a taxi, there are still some options available. Firstly, try to negotiate with the driver and explain your situation. They may be willing to accept a promise to pay later or allow you to pay via a mobile app. If the driver is unwilling to accommodate your situation, you could try asking nearby businesses or individuals for a loan or assistance. Alternatively, you could consider using a ridesharing service that allows payment via a mobile app. In any case, it's important to be respectful and grateful for any assistance you receive. Remember, it's never a good idea to try and skip out on paying for a ride, as this can damage your reputation and could result in legal consequences.*

The responses are appropriate; more importantly, the model recognizes the preconditions under which one could taking a taxi.

These examples suggest that GPT-3 could generate script descriptions in a flexible manner. We also repeat the experiment with 5 random scenarios from MCScript. We do not spot any issue in all generations of these scenarios.

Events and Temporal Order

Now we query about script events. We note that this query is nontrivial. Although GPT-3 exhibits, to a certain level, capabilities of understanding task descriptions, we do not expect the model to show enough understanding of the term *event* as we use in this thesis, because this sense differs from the senses it represents in most web texts or the original training data of GPT-3. As the most straightforward alternative, we ask for the *steps* of the scenarios.

(a) Prompt: *what are the steps of changing a light bulb? (in succinct language)*

\>> *1.Turn off the power. 2.Let the bulb cool down. 3.Remove any coverings. 4.Unscrew the old bulb. 5.Dispose of it properly. 6.Insert and screw in the new bulb. 7.Replace any coverings. 8.Turn on the power and test the light.*

(b) Prompt: *what are the steps of obtaining a phd? (in succinct language)*

\>> *1.Choose a field of study and research interest. 2.Obtain a bachelor's degree in a related discipline. 3.Apply and gain acceptance into a doctoral program. 4.Complete coursework and pass qualifying exams. 5.Develop a research proposal and get it approved. 6.Conduct original research under the guidance of a supervisor. 7.Write a dissertation presenting your research findings. 8.Defend your dissertation in a formal oral examination. 9.Publish research papers in relevant journals. 10.Obtain approval from your committee and the institution. 11.Receive your doctoral degree and graduate.*

These lists are correct. Now, we query the temporal order among these events.

(c) Prompt: *When changing a light bulb, do we unscrew the old bulb first, or insert the new bulb first? The answer is*

\>> *to unscrew the old bulb first.*

(d) Prompt: *When changing a light bulb, do we unscrew the old bulb first, or turn on the power first? The answer is*

\>> *to unscrew the old bulb first and then to turn on the power.*

(e) Prompt: *When changing a light bulb, do we unscrew the old bulb first, or turn off the power first?*

\>> *You should turn off the power before changing the light bulb.*

(f) Prompt: *when obtaining a phd, do we publish research papers first or defend the dissertation first? (in one sentence)*

\>> *Typically, in the process of obtaining a Ph.D., research papers are published before defending the dissertation.*

All of these responses are correct. It seems that the simplest way of prompting could give us a reasonable set of events with a correct temporal order[3].

Participants

(a) Prompt: *List all entities that could possibly be involved when taking a taxi, in succinct language.*

\>> *Passenger, driver, taxi company, taxi dispatch center, local transportation authority, payment processing company.*

(b) Prompt: *List all entities that could possibly be involved when fixing a flat tire, in succinct language.*

\>> *Vehicle owner, tire manufacturer, tire repair shop, mechanic, roadside assistance service.*

(c) Prompt: *List all entities that could possibly be involved when taking a flight, in succinct language.*

\>> *Passenger, airline company, airport authority, air traffic control, immigration and customs authorities, baggage handling company, aircraft manufacturer, maintenance and repair company, security personnel, ground handling company, air traffic regulatory body.*

With (a)-(c) we query the model for script participants. Generations are correct, but we find some script participants missing, such as tire in fixing a flat tire, taxi in taking a taxi and aircraft in taking a flight. These are crucial participants that should not be missed out on. However, missing these participants is probably due to the limitation of GPT-3 understanding *task descriptions*, instead of encoding an insufficient amount of script knowledge. The term *Participant* has a specific interpretation in the context of script knowledge, which deviates from its most common sense. The model does not receive specific training for terminologies, nor is *participant* so frequent as a specific terminology that a language model learns it anyways. Therefore, querying *participants* will only give us human participants in the scenarios. *Entity* is the best replacement we find, but clearly the model tends to interpret it as humans or organizations.

Note that all our queries are performed unsupervised. However, the model could be guided by a handful of examples. Again, the purpose of these examples is to

---

[3] These verbal phrases are not precisely script events, as the latter are supposed to abstract away from surface texts. But for the sake of script acquisition, they do allow us to generate temporal script graphs.

imporve task description, instead of injecting script knowledge.

(a) Prompt: *The participants of going grocery shopping includes: customer, shop, shopping cart, grocery, shopping list, cashier, money, shopping bag, checkout counter. What are the participants of taking a taxi? (in succinct language)*

>> *The participants of taking a taxi include: passenger(s), taxi driver, taxi company,* ***taxi meter****, taxi dispatch, GPS/navigation system, payment method,* ***taxi cab****, and traffic/road conditions.*

(b) Prompt: *The participants of going grocery shopping includes: customer, shop, shopping cart, grocery, shopping list, cashier, money, shopping bag, checkout counter. What are the participants of taking a flight? (in succinct language)*

>> *The participants of taking a flight include: passenger(s), airline,* ***aircraft****,* ***pilot****, flight attendant(s), air traffic control,* ***airport****,* ***luggage****,* ***boarding pass****, security checkpoint, and immigration/-customs (for international flights).*

(c) Prompt: *The participants of going grocery shopping includes: customer, shop, shopping cart, grocery, shopping list, cashier, money, shopping bag, checkout counter. What are the participants of fixing a flat tire? (in succinct language)*

>> *The participants of fixing a flat tire include: vehicle owner/driver,* ***flat tire****,* ***spare tire****, jack, lug wrench, tire pressure gauge, and potentially a roadside assistance service.*

The model is now much better at listing script participants that are usually (i.e. in most web texts) not referred to as an *entity*. The answers are still less than satisfactory, but given the learning capacity the model exhibits to capture new concepts from one single example, it is promising to find a description of detecting script participants that yields fine performance.

Summary

7.3 GPT-3 as a Script Aware Story Generation System

In the previous section we query GPT-3 for key elements of script knowledge, including events, participants, scenario descriptions, and temporal orders and see impressive performance. In this section, we evaluate GPT-3 on a downstream task of script knowledge, that is, story generation. We exploit specific prompts to construct a story generation agent out of querying GPT-3, which takes the scenario and an *agenda* as input, and produces script stories corresponding to the agenda. In Chapter 3 we show that script knowledge allows us to establish the global coherence of these stories. However, what we do here is completely unsupervised, which means it is entirely up to GPT-3 to provide script knowledge. We generate stories for the test scenarios of MCScript and evaluate this story generation system in the same way as we did in Chapters 3, and compare it to other story generation systems that have different sources of script knowledge.

Story Generation from Agendas with GPT-3

There are various possible ways to construct a story generation system out of GPT-3. Our preferences include the following. First, for comparison purposes, it must accept the same type of input as the story generation systems that we discuss in Chapters 3 and 6. That is, a scenario and an agenda. Second, it should not receive any supervision, so that GPT-3 acts as the sole source of script knowledge. Third, to ensure comparability, the generated story should fit what our human evaluation pipeline is meant for. Specifically, the language should not be too complex, and the length of the stories should not deviate away from that out of the previous generation systems and InScript itself.

Empirically, we find that the following pattern works well for our intentions. The prompt is simple and the generated story is in good correspondence with the agendas.

Prompt: *On changing a light bulb.*
Agenda: bulb breaks, climb on ladder, turn light off, screw new bulb in, light works
story:

\>> *I was in my kitchen when I noticed that the light had* ***gone*** *out. I got a ladder and* ***climbed*** *up to change the bulb. I unscrewed the old bulb and* ***screwed*** *in the new one. The light* ***came*** *on and everything was back to normal.*

Prompt: *On reading a story to a child.*
Agenda: let child choose story, turn page, read, child reads along, put book away
story:

\>> *The mother and child snuggled together on the couch, the mother with a book in her hand. "What story would you like me to read to you tonight?" she asked her child. The child* ***pointed*** *to a book on the shelf. The mother took it down and handed it to the child. The child* ***turned*** *to the first page and the mother began to* ***read****. The child* ***followed*** *along, pointing to the words as the mother read. Occasionally, the child would ask a question about the story. When the story was finished, the child* ***put*** *the book away and snuggled up against the mother. "That was a great story," the mother said. "I"m glad you chose it."*

Agenda Generation

Now we show how to generate script agendas with GPT-3. The agendas as input to our story generation systems in Chapters 3 and 6 are generated by a language model of script events estimated with the respective training sets. An important aspect of these agendas is that they encode script knowledge obtained from the training set. For the purpose of testing GPT-3 on how much script knowledge it encodes, we cannot feed these agendas to GPT-3 for story generation. Instead, we need to acquire such agendas from GPT-3.

Prompt: *generate an agenda for answering a phone (in concise phrases)*

\>> *Agenda for Answering a Phone:*
1.Ringing - Hear the phone ring and prepare to answer.
2.Greeting - Answer the phone with a friendly and clear greeting.
3.Identify Yourself - State your name or the name of the organization you represent.
4.Inquire about Caller's Identity - Politely ask who is calling.
5.Active Listening - Pay attention to the caller's response and actively listen.
6.Respond to Caller - Provide a prompt and courteous reply based on the caller's needs.
7.Offer Assistance - Ask how you can assist or address the caller's concerns.
8.Gather Information - Request necessary details or information from the caller.
9.Take Notes - Record relevant information during the conversation.
10.Provide Solutions - Offer appropriate solutions or options to address the caller's needs.
11.Confirm Understanding - Ensure that both parties have a clear understanding of the conversation.
12.Thank the Caller - Express gratitude for the caller's time and cooperation.
13.Farewell - End the conversation politely and wish the caller a good day.
14.Follow-Up Actions - Take note of any follow-up actions required on your part.
15.Disconnect - Hang up the phone after the conversation concludes.
Remember to adapt the agenda based on the nature of the call and any specific guidelines or procedures provided by your organization.

If we directly query GPT-3 for an agenda, it gives fine responses (see above). However, simply using such agendas as inputs to our querying-GPT-based story generation system would raise another concern. That is, the stability of human evaluation. Our plan is to evaluate this story generation system with the human evaluation schema we develop in Chapter 3, and compare the results to those of other automatic story generation systems. To this end, stories from different generation systems will be mixed up, and a participant will receive random stories from the pool. The variable we control is the origin of these stories, i.e. the system that generates it. The entire set of stories is expected to be somewhat homogeneous in all other aspects: These stories are on the same set of scenarios for each generation system; they are generated by story generation systems that consume roughly the same budget of computational resources when applicable; all stories admit simple language, etc. However, if we use the agendas we get from direct queries like the example above, we will incur another variable, that is, the set of events. The set of events for all other story generation systems is that induced by the MCScript annotations[4], but the set of events included in an agenda produced by GPT-3 could be quite arbitrary. As the subjects of the evaluation need to decide whether an event is realized in the stories, the difference in the names of the events matter in the process.

To address this issue, we do the following post-processing for the GPT-3 generated agendas, to reduce its set of events to that of MCScript. Basically, we query GPT-3 for an agenda, and only keep the items that correspond to some event type that exists in the annotations of MCScript. This judgement is made by the author. See below.

```
Input: Scenario $s$ from the test set of MCScript, and $\{m_i : i \leq l\}$, the
       set of events corresponding to $s$ in the annotations; sequence of
       events $\mathcal{A} \leftarrow \emptyset$.

  1. Query GPT-3 with $s$ to generate agenda $\{e_i : i \leq n\}$.

  2. For $i$ from 1 to $n$:

     if exists $j$ s.t. process $e_i$ is contained in $m_j$:

     $\mathcal{A} \leftarrow \mathcal{A}m_j$

  3. Output $\mathcal{A}$
```

We are now set to query GPT-3 for script stories to compare with the other story generation systems. We generate 4 stories per scenario for the evaluation.

Prompt: *generate an agenda for reading a story to a child (in concise phrases)*

\>> *1.Introduction and Greeting*
2.Select a Suitable Storybook
3.Settle in a Comfortable Reading Space
4.Engage the Child's Attention with an Opening Question or Preview
5.Read Aloud, Using Clear and Expressive Voice
6.Pause Occasionally to Ask Questions or Encourage Participation
7.Use Animated Facial Expressions and Gestures to Enhance the Story
8.Explain Difficult Words or Concepts as Necessary
9.Encourage Imagination and Visualization
10.Discuss the Story and Its Themes After Reading
11.Relate the Story to Real-Life Experiences, if Applicable
12.Answer the Child's Questions and Encourage Reflection
13.Express Appreciation for the Child's Listening and Participation
14.Conclude with a Farewell and Invitation to Read Again

Post-processed: *1.*
2.choose story
3.sit down
4.
5.read
6.child reads along
7.read
8.
9.
10.talk about story
11.talk about story
12.talk about story
13.finish reading
14.finish reading

Figure 7.3: An example agenda from GPT-3 and its post-processed version.

Source	Coverage	Syntax	Global Coh.	Local Coh.	Relevance	Informativeness
No S. Knowledge	3%	.25	.07	.24	.07	.07
Zero-shot	50%	**.56**	**.57**	**.54**	**.63**	**.59**
Annotated	**63%**	.43	.49	.42	.59	.51
GPT-3	83%	.91	.86	.91	.87	.85

All differences are statistically significant according to independent T-test at $\alpha = 0.05$.

Table 7.1: Results from human evaluation. All metrics are ranged $[0, 1]$.

Evaluation

Table 7.1 gives the results of the evaluation to compare with the quality of the stories generated according to various sources of script knowledge. The table is an extension of Table 6.7 from the previous chapter.

We could see that the performance of `GPT-3` is quite impressive. It outperforms all other systems by a large margin in every metric. The text is fluent and coherent; the agendas are well-covered. For the task of story generation, using GPT-3 as the source of script knowledge and the story generation system outperforms the combination of explicit script knowledge and a specialized script story generation model.

7.4 Summary

In this chapter, we mainly evaluate large language models, represented by OpenAI's GPT-3, as a source of script knowledge. Our evaluation comprises two pespectives. First, we examine whether GPT-3 could explicitly spell out key elements of script knowledge (Section 7.2). Without any supervision, GPT-3 could spell out script descriptions of fine quality. Directly querying GPT-3 for script events and participants is nontrivial as susitable task descriptions need to be developed, because *script events* and *script participants* as terminologies have specific senses. However, our preliminary inspections suggest that this should be feasible. Second, we leave aside explicit script knowledge and evaluate GPT-3 as a script-knowledge-aware agent.

---

[4] This also holds for the generation system based on script knowledge from zero-shot learning, on a slightly difference sense. In particular, the event clusters from zero-shot learning do not have event names, but we correspond the event clusters to the original MCScript events during the evaluation of Hungarian F1, when we create a mapping from the zero-shot event clusters to the MCScript events.

We wrap up GPT-3 as a script story generation system and compare its performance against the ones we train in Chapter 6, which is based on script knowledge from different sources. One of these models is trained according to script knowledge we acquire automatically, from zero-shot learning, whereas the other is trained from hand-annotated script knowledge. According to human evaluation, GPT-3 outperforms these systems by a large margin, suggesting GPT-3's high potential as a script-knowledge-aware agent.

According to our inspections, in the current state of research, it seems very difficult that NLP applications trained with explicit script knowledge outperform GPT-3. Not to mention that the latter does not require much supervision. However, we do not know enough to provide a confident answer whether the existence of large language models renders the enterprise of explicit script acquisition unnecessary. There are specific categories of common sense knowledge that large language models cannot handle smoothly (see Figure 7.4), but we are yet to find a counterexample for script knowledge.

Our observations are consistent with recent works of the community on acquiring script knowledge from pretrained language models. Sancheti and Rudinger (2022) report that event sequence descriptions generated by GPT-2 on a zero shot manner do not admit fine quality. However, with a few examples and some post-processing, the quality could be greatly improved. Li et al. (2023) was able to collect script events and arrange them in a hierarchical manner, indicating which events could be expanded to multiple events, resulting in a finer granularity.

(a) Prompt: *(1+(2*3))/2+1*

>> *5*

(b) Prompt: *333*555*

>> *183325*

(c) Prompt: *How many prime numbers there are below 100?*

>> *25*
How many prime numbers are there below 102?

>> *There are 25 prime numbers below 102.*

(d) Prompt: *Biden sold Alaska to Russia. How many states are there in the US now?*

>> *There are 50 states in the US.*

Figure 7.4: Examples where common sense knowledge is needed, but GPT-3 fail to deliver correct responses.

Conclusion

In this thesis, we develop an approach to acquire explicit script knowledge at a lower cost.

In Chapter 3, we prove that script knowledge enables story generation systems to perform better content selection. In particular, a story generation system could include richer detail in the generated stories in an appropriate manner by referring to script knowledge of the scenario of interest, as it gives clear evidence about the progress of the generation.

We propose a high performance script parser in Chapter 4 that could parse script events and participants simultaneously. In Chapter 5, we develop a zero shot script parser that could perform script acquisition given script stories on unknwon scenarios. Its performance is on par with that of a previous supervised parser. The zero-shot parser automates the human annotation process in the creation of corpora like InScript, which significantly reduces the cost of script acquisition. In Chapter 6, we evalute the script knowledge we could acquire from the zero-shot parser. On MCScript, whose texts are similar to that of InScript, the script knowledge we acquire delivers fine performance in both intrinsic evaluation (in Hungarian F1) and extrisic evaluation (by accommodating the training of a story generation system). In summary, it is now possible to perform script acquisition of unseen scenarios on unannotated, InScript style script stories. That means, the vast majority of the cost of creating InScript, which comes from munual annotation, could be evaded.

With the development of large pretrained language models, it is reasonable to anticipate that these models encode script knowledge. We explore in Chapter 7 what this implies for script knowledge acquisition. In terms of explicit script knowledge, it seems that GPT-3 is able to directly spell out key elements of script knowledge, such as events and their temporal order. Another possibility is to construct NLP applications that behave in a script-knowledge-aware manner, without explicitly stating how each script scenario looks like. We construct a story generation system by prompting which outperforms all other story generation systems we trained on different sources of explicit script knowledge. In conclusion, in the current state of research, large language models seem to be the better option when it comes to performance. It remains open whether pretrained language models are the future of script acquisition. Yuan et al. (2023) collects *CoScript*, a dataset that contains specific goals and the stps to achieve that goal. For example, a plan of making a *chocolate* cake. The dataset is of fine quality, while the data is collected by prompting InstructGPT with some post processing.

References

AlKhamissi, B., Li, M., Celikyilmaz, A., Diab, M., and Ghazvininejad, M. (2022). A review on language models as knowledge bases. *arXiv preprint arXiv:2204.06031*.

Banerjee, P. and Baral, C. (2020). Self-supervised knowledge triplet learning for zero-shot question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 151–162, Online. Association for Computational Linguistics.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bojar, O. and Tamchyna, A. (2011). Improving translation model by monolingual data. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 330–336, Edinburgh, Scotland. Association for Computational Linguistics.

Bouraoui, Z., Camacho-Collados, J., and Schockaert, S. (2020). Inducing relational knowledge from bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7456–7463.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Burton, K. R., Java, A., and Soboroff, I. (2009). The icwsm 2009 spinn3r dataset.

Busemann, S. and Horacek, H. (1998). A flexible shallow approach to text generation. In *Natural Language Generation*.

Chambers, N. and Jurafsky, D. (2008). Unsupervised learning of narrative event chains. *Proceedings of ACL-08: HLT*, pages 789–797.

Chambers, N. and Jurafsky, D. (2009). Unsupervised learning of narrative schemas and their participants. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 602–610. Association for Computational Linguistics.

Chang, M.-W., Ratinov, L.-A., Roth, D., and Srikumar, V. (2008). Importance of semantic representation: Dataless classification. In *Aaai*, volume 2, pages 830–835.

Cho, K., van Merriënboer, B., Guulçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Dong, X. and Shen, J. (2018). Triplet loss in siamese network for object tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 459–474.

Duan, X., Yin, M., Zhang, M., Chen, B., and Luo, W. (2019). Zero-shot cross-lingual abstractive sentence summarization through teaching generation and attention. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3162–3172, Florence, Italy. Association for Computational Linguistics.

Fan, A., Lewis, M., and Dauphin, Y. (2018). Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898.

Fan, A., Lewis, M., and Dauphin, Y. (2019). Strategies for structuring story generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2650–2660.

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database.* Bradford Books.

Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M., and Zettlemoyer, L. S. (2017). Allennlp: A deep semantic natural language processing platform.

Gordon, A. S. and Swanson, R. (2009). Identifying personal stories in millions of weblog entries.

Granroth-Wilding, M. and Clark, S. (2016). What happens next? event prediction using a compositional neural network model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. In *ACL (1)*.

Jans, B., Bethard, S., Vulić, I., and Moens, M. F. (2012). Skip n-grams and ranking functions for predicting script events. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 336–344, Avignon, France. Association for Computational Linguistics.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Kenton, J. D. M.-W. C. and Toutanova, L. K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Kiddon, C., Zettlemoyer, L., and Choi, Y. (2016). Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR (Poster)*.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Lampert, C. H., Nickisch, H., and Harmeling, S. (2009). Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE conference on computer vision and pattern recognition*, pages 951–958. IEEE.

Lee, I.-T., Pacheco, M. L., and Goldwasser, D. (2020). Weakly-supervised modeling of contextualized event embedding for discourse relations. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4962–4972, Online. Association for Computational Linguistics.

Levesque, H. J., Davis, E., and Morgenstern, L. (2012). The Winograd Schema Challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, pages 552–561. AAAI Press, Rome, Italy.

Li, J., Galley, M., Brockett, C., Gao, J., and Dolan, B. (2016). A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119, San Diego, California. Association for Computational Linguistics.

Li, S., Zhao, R., Li, M., Ji, H., Callison-Burch, C., and Han, J. (2023). Opendomain hierarchical event schema induction by incremental prompting and verification.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2020). Ro{bert}a: A robustly optimized {bert} pretraining approach.

Lukasová, A. (1979). Hierarchical agglomerative clustering procedure. *Pattern Recognition*, 11(5-6):365–381.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Madaan, A., Zhou, S., Alon, U., Yang, Y., and Neubig, G. (2022). Language models of code are few-shot commonsense learners. *arXiv preprint arXiv:2210.07128*.

Martin, L. J., Ammanabrolu, P., Wang, X., Hancock, W., Singh, S., Harrison, B., and Riedl, M. O. (2018). Event representations for automated story generation with deep neural nets. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Meehan, J. R. (1976). The metanovel: writing stories by computer. Technical report, YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE.

Mei, H., UChicago, T., Bansal, M., and Walter, M. R. (2016). What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *Proceedings of NAACL-HLT*, pages 720–730.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Modi, A., Anikina, T., Ostermann, S., and Pinkal, M. (2016). Inscript: Narrative texts annotated with script information. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3485–3493.

Modi, A. and Titov, I. (2014). Inducing neural models of script knowledge. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 49–57.

Modi, A., Titov, I., Demberg, V., Sayeed, A., and Pinkal, M. (2017). Modeling semantic expectation: Using script knowledge for referent prediction. *Transactions of the Association for Computational Linguistics*, 5:31–44.

Mostafazadeh, N., Chambers, N., He, X., Parikh, D., Batra, D., Vanderwende, L., Kohli, P., and Allen, J. (2016). A corpus and evaluation framework for deeper understanding of commonsense stories. *arXiv preprint arXiv:1604.01696*.

Ostermann, S., Modi, A., Roth, M., Thater, S., and Pinkal, M. (2018a). Mcscript: A novel dataset for assessing machine comprehension using script knowledge. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Ostermann, S., Roth, M., Modi, A., Thater, S., and Pinkal, M. (2018b). SemEval-2018 task 11: Machine comprehension using commonsense knowledge. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 747–757, New Orleans, Louisiana. Association for Computational Linguistics.

Ostermann, S., Roth, M., and Pinkal, M. (2019). Mcscript2. 0: A machine comprehension corpus focused on script events and participants. In *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (* SEM 2019)*, pages 103–117.

Ostermann, S., Roth, M., Thater, S., and Pinkal, M. (2017). Aligning script events with narrative texts. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*, pages 128–134.

Pearl, J. et al. (2000). Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, 19(2).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P., Bakhtin, A., Wu, Y., and Miller, A. (2019). Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473.

Pham, N.-Q., Niehues, J., Ha, T.-L., and Waibel, A. (2019). Improving zero-shot translation with language-independent constraints. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 13–23, Florence, Italy. Association for Computational Linguistics.

Philip, J., Berard, A., Gallé, M., and Besacier, L. (2020). Monolingual adapters for zero-shot neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4465–4470, Online. Association for Computational Linguistics.

Pichotta, K. and Mooney, R. (2014). Statistical script learning with multi-argument events. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 220–229.

Pichotta, K. and Mooney, R. (2016). Learning statistical scripts with lstm recurrent neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Pusse, F., Sayeed, A., and Demberg, V. (2016). Lingoturk: managing crowd-sourced tasks for psycholinguistics. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 57–61.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Rahman, A. and Ng, V. (2012). Resolving complex cases of definite pronouns: the winograd schema challenge. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 777–789.

Regneri, M., Koller, A., Ruppenhofer, J., and Pinkal, M. (2011). Learning script participants from unlabeled data. In *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pages 463–470.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA.

Riedl, M. O. and Young, R. M. (2010). Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*, 39:217–268.

Rudinger, R., Rastogi, P., Ferraro, F., and Van Durme, B. (2015). Script induction as language modeling. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1681–1686, Lisbon, Portugal. Association for Computational Linguistics.

Ruppenhofer, J., Ellsworth, M., Schwarzer-Petruck, M., Johnson, C. R., and Scheffczyk, J. (2016). Framenet ii: Extended theory and practice. Technical report, International Computer Science Institute.

Sak, H., Senior, A. W., and Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition.

Sakaguchi, K., Bhagavatula, C., Le Bras, R., Tandon, N., Clark, P., and Choi, Y. (2021). proscript: Partially ordered scripts generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2138–2149.

Sancheti, A. and Rudinger, R. (2022). What do large language models learn about scripts? In *Proceedings of the 11th Joint Conference on Lexical and Computational Semantics*, pages 1–11.

Schank, R. C. and Abelson, R. P. (1977). *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures.* Psychology Press.

Schick, T. and Schütze, H. (2021). Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269.

Sennrich, R., Haddow, B., and Birch, A. (2016). Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Stymne, S., de Lhoneux, M., Smith, A., and Nivre, J. (2018). Parser training with heterogeneous treebanks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 619–625, Melbourne, Australia. Association for Computational Linguistics.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wanzare, L., Zarcone, A., Thater, S., and Pinkal, M. (2017). Inducing script structure from crowdsourced event descriptions via semi-supervised clustering. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pages 1–11.

Wanzare, L. D., Zarcone, A., Thater, S., and Pinkal, M. (2016). Descript: A crowdsourced database of event sequence descriptions for the acquisition of high-quality script knowledge. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3494–3501.

Wanzare, L. D. A., Roth, M., and Pinkal, M. (2019). Detecting everyday scenarios in narrative texts. In *Proceedings of the Second Workshop on Storytelling*, pages 90–106, Florence, Italy. Association for Computational Linguistics.

Weber, N., Rudinger, R., and Van Durme, B. (2020). Causal inference of script knowledge. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7583–7596, Online. Association for Computational Linguistics.

Weber, N., Shekhar, L., Balasubramanian, N., and Chambers, N. (2018). Hierarchical quantized representations for script generation. In *Proceedings of the*

2018 Conference on Empirical Methods in Natural Language Processing, pages 3783–3792.

Wen, T.-H., Gašic, M., Kim, D., Mrkšic, N., Su, P.-H., Vandyke, D., and Young, S. (2015). Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. In *16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 275.

Xie, Q., Dai, Z., Hovy, E., Luong, T., and Le, Q. (2020). Unsupervised data augmentation for consistency training. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6256–6268. Curran Associates, Inc.

Xue, N., Ng, H. T., Pradhan, S., Rutherford, A., Webber, B., Wang, C., and Wang, H. (2016). CoNLL 2016 shared task on multilingual shallow discourse parsing. In *Proceedings of the CoNLL-16 shared task*, pages 1–19, Berlin, Germany. Association for Computational Linguistics.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763.

Yin, W., Hay, J., and Roth, D. (2019). Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3914–3923, Hong Kong, China. Association for Computational Linguistics.

Yuan, S., Chen, J., Fu, Z., Ge, X., Shah, S., Jankowski, C. R., Yang, D., and Xiao, Y. (2023). Distilling script knowledge from large language models for constrained language planning. *arXiv e-prints*, pages arXiv–2305.

Zhai, F., Demberg, V., Shkadzko, P., Shi, W., and Sayeed, A. (2019a). A hybrid model for globally coherent story generation. In *Proceedings of the Second Workshop on Storytelling*, pages 34–45, Florence, Italy. Association for Computational Linguistics.

Zhai, F., Demberg, V., Shkadzko, P., Shi, W., and Sayeed, A. (2019b). A hybrid model for globally coherent story generation. In *Proceedings of the Second Workshop on Storytelling*, pages 34–45.

Zhai, F., Škrjanec, I., and Koller, A. (2021). Script parsing with hierarchical sequence modelling. In *Proceedings of* SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, pages 195–201.

Zhang, B., Williams, P., Titov, I., and Sennrich, R. (2020). Improving massively multilingual neural machine translation and zero-shot translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639, Online. Association for Computational Linguistics.