VRIJE
UNIVERSITEIT
BRUSSEL

Master thesis submitted in partial fulfilment of the requirements for the degree of Master of Science In de Ingenieurswetenschappen: Computerwetenschappen

A BENCHMARK FOR RECIPE UNDERSTANDING IN AUTONOMOUS AGENTS

Creation and Analysis of a New Recipe Execution Benchmark

Robin De Haes

2022-2023

Promotor: Prof. Dr. Paul Van Eecke
Advisor: Dr. Jens Nevens
Science and Bio-Engineering Sciences

VRIJE
UNIVERSITEIT
BRUSSEL

Proefschrift ingediend met het oog op het behalen van de graad van Master of Science In de Ingenieurswetenschappen: Computerwetenschappen

EEN BENCHMARK VOOR HET BEGRIJPEN VAN RECEPTEN DOOR AUTONOME AGENTEN

Creatie en analyse van een nieuwe benchmark voor het uitvoeren van recepten

Robin De Haes

2022-2023

Promotor: Prof. Dr. Paul Van Eecke
Advisor: Dr. Jens Nevens
Wetenschappen en Bio-ingenieurswetenschappen

Abstract

Procedural text is text composed of instructions on how to perform a specific task. Understanding such text requires insight in the impact of events on the world, which is a major component of intelligence. Benchmarks that support the development of artificial intelligence approaches towards achieving this capability often use recipes as a source of procedural text, because recipes are abundantly available and have consistent tasks and tools. Moreover, deep recipe understanding could lead to many practical applications such as the robotisation of everyday cooking. Despite the existence of these recipe benchmarks, however, the performance of robots is still limited when it comes to understanding an arbitrary recipe enough to allow execution. Current recipe understanding benchmarks have failed to achieve widespread adoption and lack important benchmark properties, which can slow down progress. They have no clear evaluation methods that could transfer results to the real world; they overly promote specific types of models and they miss transparency regarding benchmark applicability and design. In this thesis, we therefore developed a new benchmark, called the MUHAI Recipe Execution Benchmark, with all choices, properties and general usage being transparently analyzed and documented. In particular, we have first created the MUHAI Cooking Language which is a graph-based, machine-readable representation language for recipe execution. A test set with gold standard recipe annotations in this language is then obtained through a data curation process. To avoid overfitting and overpromotion of a particular approach, only test data is curated to separate model development from model evaluation. To perform model evaluation, a symbolic simulator is developed that measures performance using both simulation-based and non-simulation-based metrics. This combination allows for multiperspective performance estimates which optimizes transferability to real-world utility. Smatch, a commonly used semantic graph comparison tool, is included because of its high adoption rate, while simulation-based metrics are included because of their specificity towards gauging recipe execution abilities. Developed simulation-based metrics are 'goal-condition success' which measures how many required goal-conditions have been traversed during recipe execution, 'dish approximation score' which estimates similarity between two prepared food products and 'recipe execution time' which measures efficiency. Whether the developed benchmark will achieve community-wide adoption and lead to significant advances in natural language understanding can only be determined in the future. Nevertheless, its transparent design makes it useful for developing recipe understanding approaches and could even serve as inspiration for new benchmarks.

Samenvatting

Procedurele tekst is tekst dat is opgebouwd uit instructies om een specifieke taak uit te voeren. Zulke tekst begrijpen vereist inzicht in de impact van gebeurtenissen op de wereld, wat een belangrijke component is van intelligentie. Benchmarks voor de ontwikkeling van artificiële intelligentie technieken voor de verwerving van dit vermogen gebruiken vaak recepten als tekstbron, omdat recepten rijkelijk beschikbaar zijn en consistente taken en gereedschap bevatten. Bovendien zou diepgaand begrip van recepten kunnen leiden tot diverse toepassingen, zoals de robotisering van het alledaagse koken. Ondanks het bestaan van deze benchmarks zijn de prestaties van robots echter nog steeds beperkt als het aankomt op recepten voldoende begrijpen om ze effectief te kunnen uitvoeren. Huidige benchmarks voor receptontleding bereikten geen algemene ingebruikname en hebben enkele tekorten in kwaliteit die vooruitgang kunnen vertragen. Duidelijke evaluatiemethodes zijn afwezig; er is vaak overpromotie van middelen-intensieve modellen en er is weinig transparantie wat betreft ontwerpkeuzes en toepassingsgebied van de benchmark. Daarom hebben we in deze thesis een nieuwe benchmark ontwikkeld, namelijk de MUHAI Recipe Execution Benchmark, waarbij alle keuzes, eigenschappen en algemeen gebruik openlijk geanalyseerd en gedocumenteerd zijn. Meer bepaald hebben we eerst de MUHAI Cooking Language gecreëerd die een op grafen gebaseerde machineleesbare representatietaal is voor receptuitvoering. Een testset met annotaties van recepten is vervolgens verworven via een datacuratieproces. Om overfitting en overpromotie van specifieke aanpakken te vermijden, wordt er enkel testdata geleverd om modelontwikkeling en -evaluatie te scheiden. Om modellen te evalueren is er een symbolische simulator ontwikkeld die prestaties meet via diverse metrieken, waarvan de meeste op simulatie gebaseerd zijn. Deze combinatie van metrieken laat multiperspectieve schattingen toe wat hun scores beter overdraagbaar maakt naar nut in de echte wereld. Smatch, een veelgebruikte vergelijkingstool voor semantische grafen, is geïncludeerd omwille van zijn hoge adoptiegraad, terwijl op simulatie gebaseerde metrieken zijn toegevoegd omwille van hun specificiteit wat betreft het ijken van succes in receptuitvoering. Ontwikkelde op simulatie gebaseerde metrieken omvatten 'goal-condition success' dat het doorlopen van nodige tussendoelen meet, 'dish approximation score' wat de gelijkenis tussen twee bereide voedingsproducten schat en 'recipe execution time' wat efficiëntie meet. Of de ontwikkelde benchmark tot vooruitgang zal leiden kan nog niet vastgesteld worden, maar zijn open ontwerp maakt hem hoe dan ook nuttig voor de ontwikkeling van technieken voor receptontleding en zelfs als inspiratie voor nieuwe benchmarks.

Acknowledgements

Without the guidance, advice and support I have received during my studies, I would not have been able to complete this thesis and present you with something today that I am very proud of. That is why I would like to take this opportunity to thank some people that have been particularly important to me throughout this process.

First and foremost, I would like to thank my promotor Professor Paul Van Eecke and my advisor dr. Jens Nevens. Their help has been essential in completing my thesis and especially in enhancing its quality. Together with Professor Katrien Beuls and Lara Verheyen, they also gave me my first introduction to the domain of natural language understanding during courses I took at the Vrije Universiteit Brussel. Their teaching approach sparked my interest in this challenging domain and is one of the main reasons I chose to delve deeper into it. Additionally, I would like to thank them for actually giving me the opportunity to work further on these topics in the context of my thesis and guiding me while I was taking my first real steps in creating an actual scientific contribution.

But I should not forget about my fellow students with whom I laid the initial foundations of this project during the course Artificial Intelligence Programming Paradigms at the Vrije Universiteit Brussel in 2021. We had many discussions during that course which have led to insights that have proven to be very useful during my further research. Special thanks in this regard should be given to Seppe Lampe with whom I have had many late-night exchanges of ideas during that time.

Not only my fellow students of Artificial Intellingence Programming Paradigms deserve my gratitute. Together with the VUB institution as a whole, every student I have met has made these years of master studies an amazing experience which reinstilled my joy for learning and taught me to have an open and curious yet critical mind.

Last but not least, I want to sincerely thank my parents, brothers, sisters and especially my girlfriend Camila Nalio Spricigo. Their unconditional encouragement and support when I decided to go back to university, years after starting a professional career, have been indispensable for lifting me up when motivation was low.

The initial idea behind this thesis' topic has been developed in the context of the European project MUHAI, which aims to introduce Meaning and Understanding in Human-centric Artificial Intelligence.

Contents

Chapter 1

Introduction

Natural language understanding (NLU) has captivated both academic and industrial researchers since the start of the field of artificial intelligence (AI; Jones, 1994). In order to reach general-purpose NLU a lot of research has already gone into developing methodologies and benchmarks that aim to be as general-purpose as possible (Cambria & White, 2014; Wang et al., 2019). A lot of this research focuses on declarative language use, which is the language that is generally used in news papers or everyday conversations for example to convey information by declaring facts or opinions. However, understanding procedural language, which generally consists of instructional steps for performing a specific task, is equally important both from a scientific and a practical viewpoint. From a scientific viewpoint, it can be stated that understanding the impact that certain actions or events will have on the world is a major component of intelligence. Procedural text understanding generally requires modeling and reasoning with a dynamically changing world and can therefore be seen as a representative task for estimating this ability (Henaff et al., 2017; Mishra et al., 2018). From a practical viewpoint, having agents that understand procedural language could for example progress the development of robotic workers that could execute a variety of manual tasks (J. Ribeiro et al., 2021).

The cooking domain in particular has been popular in this context. This popularity can be explained by the availability of a large amount of recipe data, a clear objective that should be reached in the end and the possibility of making some simplifying assumptions about the environment due to general consistency in objects, tools and tasks (Bollini et al., 2013; Kiddon et al., 2015; Maeta et al., 2015). Furthermore, cooking is a common and frequent activity taking place in both personal and professional environments. This means that a practical application such as a cooking robot that can follow arbitrary recipes would be able to relieve a lot of manual labor. Especially in the industrial setting, there is an increasing demand for robotic help in recent times due to changed customer perspectives after the COVID-19 pandemic and unresolved worker shortages in restaurants and food supply chains (Chuah et al., 2022).

Due to this popularity and need, there exists quite a bit of prior research related to applying AI techniques inside the cooking domain. These techniques vary from generating food recommendations (Elsweiler et al., 2017) to being able to robotically cook a dish

starting from just a recipe written in English (Beetz et al., 2011; Bollini et al., 2013). However, results of these cooking robot experiments so far have been inadequate for general use except in very standardized settings that require limited domain knowledge. Investing further resources in the different subsystems of such a cooking robot is needed. Progression is still needed in robotic manipulation systems, computer vision and also in NLU.

An approach that has proven to be successful in steering and boosting progress in diverse scientific fields is the use of benchmarks (Donoho, 2017). Therefore, throughout the years some research has already been put into creating benchmarks for recipe understanding with the aim of parsing natural language text to an executable machine-readable and mostly graph-based format (Jiang et al., 2020; Tasse & Smith, 2008; Yamakata et al., 2020). However, these benchmarks have currently failed to achieve community-wide adoption because they lack certain properties that are considered to be important for ensuring high benchmark quality.

Therefore, the development of a new recipe understanding benchmark seems useful for progressing the field of procedural language understanding with the ultimate goal being the achievement of general robotic recipe execution. Such a benchmark should reuse concepts and ideas from prior research that seem to be beneficial while improving upon components that are lacking in previously developed benchmarks.

1.1 Problem Definition

Cooking is a common human activity that has certain characteristics making it suitable for execution by an artificial agent. Although a lot of research and progress has already been made in this context, further efforts are needed and existing resources seem inadequate. Therefore, the main goal of this master thesis is the development of a new recipe understanding benchmark with a strong focus on applicability in the context of robotic execution. Explicitly, the research question studied can be stated as follows:

"How can we design a benchmark that can foster progress in the domain of natural language understanding with a focus on the deep understanding of an everyday human activity, in particular the execution of recipes written in natural language in a kitchen environment?"

Answering this question will require research into the current status of the domain of NLU and into benchmarking. Subsequently, insights gained from this research will have to be put in practice in order to develop a new benchmark that can progress the field. In other words, our research question involves finding an answer to multiple subquestions such as:

- What are the properties of a good benchmark?
- What relevant methodologies and benchmarks exist in the domain of natural language understanding for recipe execution?

- How can we practically build a benchmark that mitigates common benchmark issues, while maximally preserving their benefits?

1.2 Thesis Structure

To be able to give a clear answer to our research question, we have structured our thesis as follows. In Chapter 2, we will give an overview of relevant background knowledge that will aid in understanding the research performed in this thesis. This includes investigating the current state-of-the-art when it comes to recipe understanding and execution as well as formally defining what a benchmark is and analyzing which properties it should possess to be of high quality.

Chapter 3 then defines a new benchmark based on the insights that are gained from Chapter 2. An important point of focus for this new benchmark has been application-specificity and evaluation, with extensive effort being put into building a kitchen simulator and metrics that allow for transferring performance results to the real world. Furthermore, openness and transparency about made design choices, their benefits and their potential drawbacks have been made a priority. This is further accentuated by an in-depth analysis of the benchmark's properties at the end of this chapter.

Chapter 4 explains the newly developed benchmark from a more practical perspective by illustrating how it can be set up and used. This includes both technical setup and usage information as well as highlighting some general linguistic and extralinguistic challenges researchers will encounter while building models for recipe understanding.

Finally, in Chapter 5 we will conclude our thesis with a brief summary and discussion followed by an outlook on potential future work.

Chapter 2

Background

Understanding the made decisions and the research presented in this thesis will be facilitated by having background knowledge in two broad domains. These are the domain of recipe understanding and the domain of benchmarking. In section 2.1 we will present the current state-of-the-art when it comes to understanding recipes written in natural language. Section 2.2 will then delve deeper into the characteristics, importance, benefits and risks of benchmarking in the field of AI and natural language processing (NLP) in particular. Section 2.3 handles the intersection of the aforementioned domains by providing an overview and analysis of relevant recipe-related corpora. Finally, in section 2.4 we motivate why a new benchmark -and in extension this thesis- is useful based on the presented background information.

2.1 Understanding Recipes

Processing, interpreting and understanding natural language has garnered the interest of both researchers and industry since the late 1940s (Cambria & White, 2014; Jones, 1994). Throughout the years it evolved from slow syntax-oriented machine translation using punch cards and batch processing to real-time and more semantics-oriented approaches that try to solve a variety of tasks based on world knowledge, symbolic reasoning and statistical inference. However, many problems still remain to be solved before we can shift from mere processing to what is commonly described as true general-purpose natural language understanding (NLU; Allen, 1995). Therefore, a lot of the current research either focuses on overcoming one of these specific problems in NLU or limits itself to a more specialized, manageable domain (Cambria & White, 2014).

Our research belongs to the latter branch of research. Consequently, this background section will mainly try to provide insight into concepts, terms and prior research specifically related to understanding written recipe texts. Furthermore, the variety in prior endeavors in the field of cooking-related systems that demonstrate an understanding of recipes beyond the level of syntax is rather big. This deeper understanding can for example be shown by distilling general cooking knowledge from recipes (Oonita & Kitayama, 2020; Yamaguchi et al., 2020; Yoneda & Nadamoto, 2018), by recommending food al-

ternatives based on health, user preferences or food pairings (Elsweiler et al., 2017; Ge et al., 2015; Nishimura et al., 2022; Simas et al., 2017; Twomey et al., 2020; Yasukawa et al., 2014), by providing general culinary analysis and modifying calories, cuisine styles or difficulty levels (Harashima et al., 2020; Kazama et al., 2018; Mohammadi et al., 2020; Sajadmanesh et al., 2017; Seki & Ono, 2014), by instructional question anwering (Liu et al., 2020; Yagcioglu et al., 2018), by aligning recipe instructions with videos or images (Liu et al., 2020; Malmaud et al., 2015) or by creatively generating partial or whole recipes (K. R. Chandu et al., 2020; Kiddon et al., 2016; Lee et al., 2020; Pini et al., 2019; Salvador et al., 2019). However, this section will focus on recipe parsing and processing that might ultimately lead to a robot being able to actually execute recipes. Therefore, the presented state-of-the-art will mostly be related to research that yields a formal and adequately detailed machine-readable semantic representation of recipes after parsing.

Lastly, it is important to note that only research published in English has been considered in order to prevent translation issues. However, some of this research published in English still concerns Japanese recipes. Prior research by Yamakata et al. (2017) and Sato et al. (2016) has shown that significant structural differences can be found between English and Japanese recipes. These differences range from a distinct way of referencing intermediate results to not including specifications of certain preparation processes and having a completely different instruction flow when describing the cooking process. This can lead to some steps being omitted, added or modified during the NLU process (Yamakata et al., 2020). Nevertheless, research related to Japanese recipes has still been included as many concepts are useful for English recipes even if they cannot always be used directly as presented. We will indicate what might be different for English recipes where relevant.

In the next sections we will first provide an overview of what understanding a recipe entails and which linguistic phenomena are generally encountered during this process. After that, we will give an overview of the recipe representations and approaches that are used in the current state-of-the-art. Finally, we will also include some examples of successful robotic recipe execution.

2.1.1 Recipes as Procedural Text

Recipes can be categorized under the subdomain of natural language that is called procedural text. Procedural text distinguishes itself from other types of text by being more goal-oriented in the sense that it generally consists of a sequence of sentences representing ordered instructions that lead to achieving a certain goal state through sequential or parallel execution (Maeta et al., 2015). In contrast to general language use procedural texts also contain clearer, shorter and more imperative, domain-specific sentences with less modal aspects and viewpoint dependencies. The aforementioned characteristics make procedural text an interesting target for AI applications, because understanding this more objective text comes down to learning and executing this sequence of procedural actions while tracking the resulting states to eventually work towards one or more definable goals. These are suitable conditions for many known AI paradigms (Nishimura et al., 2020). The general popularity of cooking and subsequent large amount of available

data, i.e., recipe texts, that exist online and offline further make recipes in particular a commonly used form of procedural text in AI research (Kiddon et al., 2015).

A prototypical recipe text is generally formatted as a triple consisting of a title, an ingredient list and a list of step-by-step instructions. In some recipes, however, there might not be an explicit ingredient list as fetching and preparing ingredients is made part of the listed instructions (Kiddon et al., 2015; Yamakata et al., 2017). Both these ingredient and instruction lists can be interpreted as procedures for the creation or manipulation of objects, with the title giving some indication of the end goal that will be reached (Maeta et al., 2015; Mori, Maeta, Sasada, et al., 2014).

2.1.2 Linguistic Analysis of Recipes

As mentioned in the previous section, recipe text has some inherent qualities that facilitate processing it. Nevertheless, some linguistic phenomena are still present that could be problematic and need to be addressed to allow successful interpretation. Some of these phenomena can be seen as fairly common NLU subtasks, while others are more specifically related to recipes. In both cases, however, the approaches in solving them are generally specialized to the domain of cooking. To better understand the next sections, we will give a brief overview of the processing tasks that are frequently needed when parsing recipe text.

Word Segmentation

In contrast to English where words are generally separated by whitespace characters, some languages do not have such clear word boundaries. Japanese sentences for example do not necessarily contain explicit delimiters separating the words (Yamakata et al., 2017). In such languages a process called word segmentation generally has to be conducted before being able to perform any other NLU tasks (Nakagawa, 2004).

For processing Japanese recipes word segmentation has mostly been defined as a classification problem. Machine learning methods, such as support vector machines (SVMs), conditional random fields (CRFs) or logistic regression, can be used to predict whether a boundary between two characters is likely to exist or not (Kudo et al., 2004; Neubig et al., 2011a). For class prediction, information about surrounding characters and the presence or absence of words in available dictionaries are used as features. More recently, some specialized neural network based architectures have also proven to be useful for Japanese word segmentation outside of the cooking domain (Higashiyama et al., 2020; Kitagawa & Komachi, 2018).

Part-of-Speech Tagging

Part-of-speech (POS) tagging is a process in which tags or categories are assigned to words in order to indicate the part of speech they correspond to, i.e., to indicate the grammatical function they have in a sentence. This is a preprocessing step that is performed in many NLP applications, since further analysis can benefit from knowing this

type of information (Voutilainen, 2005). Commonly used grammatical categories in this context are adjective, adverb, conjunction, determiner, particle, noun, pronoun, numeral and verb (Petrov et al., 2012). To assign these categories, modern taggers generally use both a word's definition and contextual information, e.g., the function or morphological form of surrounding words. Such taggers could be based on handwritten rules, could be automatically derived from tagged text corpora through statistical methods or a combination of both statistical and handwritten rules could be used (Voutilainen, 2005).

Although general-purpose POS taggers seem to perform very well, with an accuracy exceeding 97% when applied to well-known benchmarks, their performance can degrade severely when applied in a new domain with differences in topic or writing style compared to the data they are trained on (Gimpel et al., 2010; Manning, 2011). Since the training data for many general-purpose POS taggers is based on news paper corpora, this can lead to problems when applying them directly to cooking recipes (Cambria & White, 2014; Chang et al., 2018). Recipes mostly contain imperative and rather concise sentences, which do not occur so frequently in well-known news paper corpora such as the Penn Treebank (Donatelli et al., 2021; Marcinkiewicz, 1994). Therefore, recipe parsing might require some additional training or modifications to such general-purpose taggers (Chang et al., 2018; Malmaud et al., 2015; Sasada et al., 2015). Furthermore, it should be noted that for languages without word boundaries word segmentation and POS tagging is often intertwined as both processes provide useful information that can be used by each other (Mori et al., 2012; Nakagawa, 2004).

Named Entity Recognition

Understanding information that is present in texts generally requires recognizing the correspondence between certain spans of words in that text and real-world named entities (NEs). This process is called named entity recognition (NER; Grishman and Sundheim, 1996). NER in a sentence can be seen as a sequence labeling problem, in which the most likely combination of NE-related tags should be assigned to a sequence of words (Li et al., 2020; Nadeau & Sekine, 2007; Yamakata et al., 2020).

To solve this sequence labeling problem a lot of general-purpose NER tools have been developed that can be used (Borthwick, 1999; Ratinov & Roth, 2009; Sang & De Meulder, 2003; Sasada et al., 2015). However, domain-specific NE definitions have also proven to be useful in facilitating NER in some cases (Abacha & Zweigenbaum, 2011; Tomori et al., 2016). In the context of recipe understanding domain-specific NEs are mostly related to concepts such as food, tool, duration, quantity, actions or state (Jiang et al., 2020; Mori, Maeta, Yamakata, & Sasada, 2014; Sasada et al., 2015; Yamakata et al., 2020). Both general-purpose and domain-specific NER tools are generally based on machine learning or deep learning algorithms (Li et al., 2020; Nadeau & Sekine, 2007). Many of these algorithms expect the input sequence of words to have been augmented with POS tags.

Anaphora Resolution

Due to the concise nature of recipe texts, expressions involving references are frequently used to avoid redundancy. Resolving such references, i.e., finding the referent that words have in common, is an important task in many NLP applications and has been studied extensively. However, this research is mostly based on declarative text while procedural anaphora resolution has been studied less extensively so far (Fang et al., 2022).

Recipes in particular seem to have some idiosyncrasies that can deviate from declarative text such as news papers or common dialogues. First, so-called zero anaphora are very common. Arguments of verbs are often elided where they are usually syntactically required, because it is assumed that the reader can imply the argument from either the context or commonsense cooking knowledge (Malmaud et al., 2014). Secondly, not only shorter synonyms or pronouns are used as a reference, but hyponyms, meronyms and metonymic generic phrases frequently occur as well (Nanba et al., 2014). This means that knowledge about part-whole relations between entities (for hyponyms and meronyms) or general characteristics of an entity (for metonyms) become important for correct resolution and interpretation. Thirdly, recipes generally include an ingredient list while instructions frequently drop determiners when referring to those ingredients (Donatelli et al., 2021). It is for example not uncommon to say 'add eggs' instead of 'add the eggs' when referring to the 'six eggs' mentioned in the ingredient list. Ensuring this connection is made can be important for being able to interpret the recipe as a whole. Lastly, it is crucial to realize that even though linguistically two expressions might have the same referent this does not have to be the case semantically. The state of the real-world entity changes during recipe execution, which means a word like 'eggs' could first be used to refer to shell eggs and later to the same eggs without their shell (Fang et al., 2022).

Resolving anaphora from a syntactical, semantic and pragmatic perspective is often seen as one of the main challenges in recipe interpretation. Therefore, we refer to section 2.1.3 for more information about methods that are used for finding correct connections in recipe texts. These vary from simple recency heuristics to the use of spanning trees, ontologies and state simulators.

Knowledge Bases & Ontologies

As mentioned in the previous section, resolving anaphora is an intrinsic part in understanding a recipe. Simple references could be solved by a dictionary approach in which words are normalized to a canonical form and then matched (Harashima & Yamada, 2018; Harashima et al., 2020; Miller, 1995). Such a normalization process could already solve many problems related to synonyms, hyponyms, meronyms and abbreviations. Additionally, it could be used to correct misspellings which might be prevalent in case the source data are user-generated recipes from an online source (Chung, 2012). However, more complicated references and zero anaphora require access to cooking knowledge for correct disambiguation. Moreover, it is not uncommon for cooking recipes to even omit certain information entirely (Fang et al., 2022). A recipe involving eggs for example will generally not explicitly mention the process of cracking eggs to obtain their contents,

but will simply state to add eggs. Quantities might not always be specified either, e.g., a statement indicating to season with salt without explicitly stating how much salt should be used. Therefore, more extensive knowledge bases and ontologies can be very useful in both recipe parsers and executors.

Such an ontology formally and explicitly specifies concepts (such as cooking actions, ingredients, tools and their states), their properties, affordances and relations between them (Gruber, 1995; Studer et al., 1998). Many broad, more general-purpose ontologies exist for knowledge acquisition, representation and reasoning (Lenat, 1995; Suchanek et al., 2008; Tenorth & Beetz, 2009) that have been applied in recipe parsing and execution. However, these are often more suited for representing declarative knowledge instead of procedural knowledge (Cambria & White, 2014). Therefore, researchers have also focused on developing and incorporating domain-specific cooking ontologies and databases (Badra et al., 2008; Cordier et al., 2009; Nanba et al., 2014; R. Ribeiro et al., 2006). As an alternative to using external ontologies and databases, some efforts have been made to address reference resolution through the use of neural network architectures in which commonsense cooking knowledge is implicitly encoded (Jiang et al., 2020; Kiddon et al., 2015). However, these approaches generally require large training datasets to get reasonable results.

2.1.3 Recipe Parsing

The procedural nature of recipes led to implicit or explicit graph-based representations being the most frequently used way of representing recipes in a structured machine-readable way (Donatelli et al., 2021). Graphs are a common representation choice for recipes because they can effectively and formally capture ingredient, tool and action semantics while also indicating dependency, part-of and other relationships between them. This allows for easier manipulation, interpretation and execution by agents (Papadopoulos et al., 2022). An illustrative example of a possible graph-based representation is given in Figure 2.1.

Despite the general prevalence of recipe graphs, the type of graph and level of detail that is used can still vary a lot among representations. Moreover, a similar remark can be made for the parsing approaches that are used to obtain these graphs from recipe texts. Supervised, semi-supervised and unsupervised approaches have been investigated with varying success. These approaches could be rule-based or rely on machine and deep learning methods.

In the following paragraphs we will give an overview of existing parsing approaches. First we will present unimodal approaches in which understanding is solely based on recipe text as will be the case in our benchmark, followed by some interesting multimodal approaches that try to enhance the parsing process by using images or videos in addition to the recipe text.

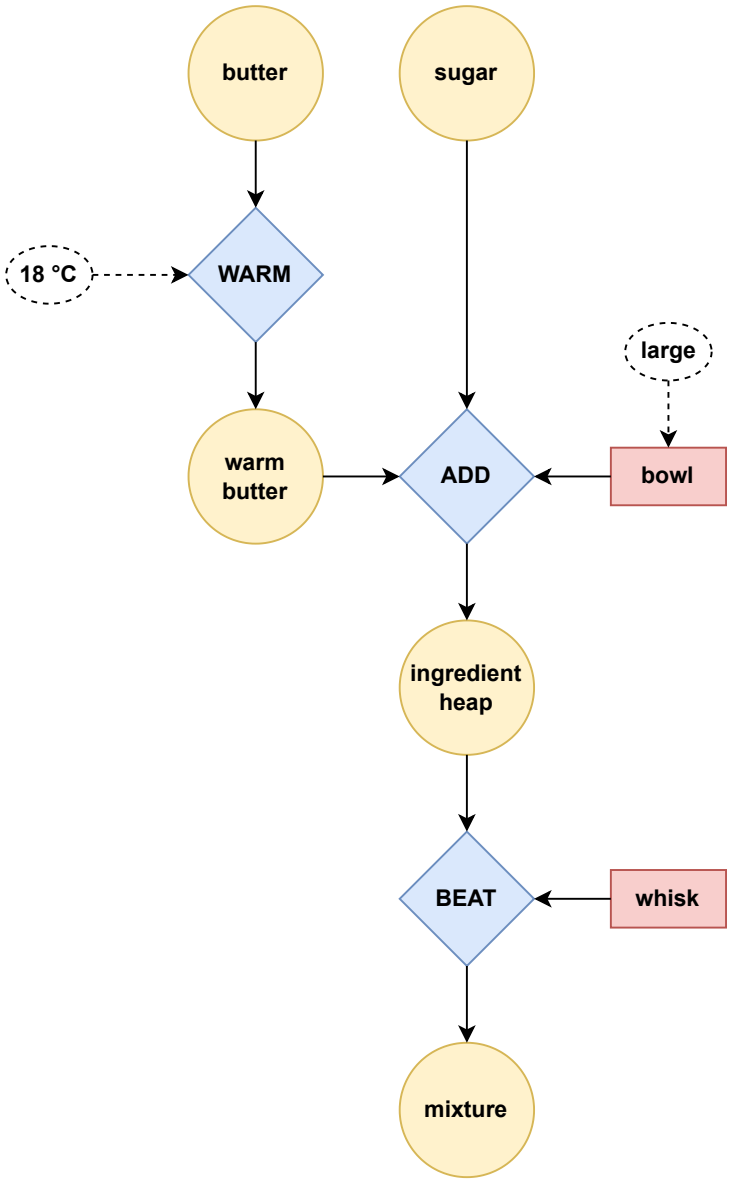

Figure 2.1: Graph representation for a recipe extract in which some warm butter and white sugar are beaten together with a whisk in a large bowl to form a mixture. Actions are displayed as diamonds, food products as circles, tools as rectangles and additional information as ellipses. We invented this graph representation solely for illustrative purposes.

Graph-based Representations

Hamada et al. (2000) performed one of the earliest works on graph-oriented recipe processing. They tried to create data flow graphs from Japanese recipes by performing a structural text analysis while incorporating cooking knowledge gathered in domain-specific dictionaries. They created these cooking dictionaries by automatically gathering keywords from cookbooks and then preliminarily categorizing them in dictionaries based on the frequency of word occurrences and co-occurrences, e.g., seasonings are expected to occur more frequently than regular ingredients. Nouns, i.e., ingredients and seasonings, and verbs, i.e., actions, are then extracted via simple matching after which a rule-based approach is used to relate verbs to their associated nouns and finally connect these noun-verb sets to form a data flow graph with ingredients and seasonings as the main data. Generalization issues, however, exist as the used vocabulary and sentence structure can be quite variable between recipes making a lot of manual corrections being needed to ensure reliable dictionary creation.

Another proposal for automatically extracting both common cooking knowledge and building a recipe graph by parsing recipe texts was made by Kiddon et al. (2015). They proposed an unsupervised machine learning approach in which two hard Expectation Maximization (EM; Dempster et al., 1977) algorithms are used on a collection of thousands of English recipes that they gathered from Allrecipes.com[1]. The first one learns a segmentation model to extract cooking actions with their respective inputs and outputs, while the second one learns a probabilistic distribution over connections between these actions. Together these models can be used to obtain the most likely action graph representations of the recipes. Common cooking knowledge is also implicitly encoded in these models as they specify for example that a 'mix' action that leads to 'dough' will probably involve an input of 'flour' and another ingredient.

Tasse and Smith (2008) developed a formal, domain-specific representation language to translate recipe statements into a set of medium-grained machine-interpretable instructions. Their Minimal Instruction Language for the Kitchen (MILK) consists of first-order logic predicates, that form an implicit graph through shared predicate arguments, extended with concepts of temporal instruction order, ingredient creation and ingredient deletion. Executing a MILK instruction leads to a change in the state of the world, i.e., the kitchen, with an ingredient or tool being modified in some way. Therefore, parsing a recipe into MILK and then keeping track of the consecutive kitchen states should allow an agent to gain insight into the cooking process. It should be noted, however, that due to the conciseness of MILK a 'do'-instruction was included to support parsing any unknown action. Such very general instructions might complicate semantic interpretation. Furthermore, Tasse and Smith (2008) actually provided the public database Carnegie Mellon University Recipe Database (CURD), containing 260 English recipes annotated in MILK, and trained some preliminary parser models on them. The parsing accuracy of these models, however, were deemed insufficient for use.

Building on CURD Jermsurawong and Habash (2015) developed a new dataset of ingredient-instruction dependency trees for recipes, called the Simplified Ingredient Merging Map in Recipes (SIMMR). SIMMR is actually less expressive than MILK as it

only focuses on modeling the flow of ingredients throughout the recipe without semantically specifying what each individual instruction means. The leaf nodes in a SIMMR tree consist of the initial list of ingredients, while internal nodes consist of recipe instructions. A hierarchical tree edge between two nodes then signifies that the ingredient or instruction output of a child node is used as input for the instruction of the parent node. Jermsurawong and Habash (2015) converted CURD to a database of SIMMR trees to then train a Linear SVM^{rank} classifier on the result in order to learn recipe parsing. For each possible edge more than a thousand features are considered to predict its existence. In contrast to the MILK parsers of Tasse and Smith (2008), the obtained accuracy of SIMMR parsing was over 90% which is adequate for many use cases. However, due to the high-level nature of the SIMMR representation these efforts are mostly useful for investigating internal recipe structure and similarity and further extensions would be needed to become a true semantic representation closer to MILK (Jermsurawong & Habash, 2015).

Jiang et al. (2020) decided to deviate from representations such as MILK that require predefining domain-specific cooking predicates. Instead, they strive to comprehend recipes by combining generic NLP subtasks in order to make their approach less limited to the cooking domain. More specifically, Jiang et al. (2020) aim to create directed acyclic graphs (DAGs) in which nodes represent recipe-related entities such as ingredients, tools, actions or intermediate results and edges denote relations between them. These relations are described through generic predicate-argument structures from PropBank (Kingsbury & Palmer, 2002), which captures general semantic concepts and roles instead of being specifically tailored to the cooking domain. After creating their own annotated dataset, consisting of the same base recipes as the SIMMR dataset (Jermsurawong & Habash, 2015), Jiang et al. (2020) experimented with neural network model configurations combining Bidirectional Long Short-Term Memory (BiLSTM) architectures (Schuster & Paliwal, 1997) with pre-trained word embedding vectors from GLoVe (Pennington et al., 2014) and BERT (Devlin et al., 2018) which is generally seen as the current state-of-the-art in many NLP applications (McCann et al., 2017; Wolf et al., 2020). Although initial results of these models seem promising, a larger training dataset seems needed to be able to further improve performance. Furthermore, it should also be noted that the generic nature of PropBank predicate-argument structures might make actual cooking execution by robotic agents more difficult due to being less tailored to that specific domain.

Approaches based on DAG representations through machine learning have also been very popular for parsing Japanese recipes. Mori et al. (2012) used DAGs as the basis for representing individual recipe sentences as work flows. They proposed a combination of machine learning approaches composed of two big phases. In the first phase a combination of pointwise prediction classifiers (Neubig et al., 2011b) was used for word segmentation and NER, followed by a pointwise syntactic analyzer (Flannery et al., 2011) to obtain a dependency tree composed of segmented word nodes with NE labels also being attached to subtrees. In the second phase a rule-based approach is then used to transform this dependency tree into a machine-readable predicate-argument structure

called a recipe work flow. This work flow proposal, however, represents each sentence in isolation which means it has no solution for linguistic issues that span multiple sentences, e.g., coreference resolution. Mori et al. (2012) acknowledged this limitation and performed follow-up research to extend their representation to a more elaborate recipe flow graph (Maeta et al., 2015; Mori, Maeta, Yamakata, & Sasada, 2014).

A recipe flow graph adds connections between all sentences in the recipe text and has a special root specifying the final dish that will be created upon recipe execution. First, Mori, Maeta, Yamakata, and Sasada (2014) created a public corpus composed of recipe texts annotated with the aforementiond flow graphs. Later, they then used this corpus to extend their prior parsing approach in order to produce interconnected recipe flow graphs instead of isolated work flows for each recipe sentence (Maeta et al., 2015). Word segmentation and NER remained, but the pointwise syntactic analyzer and subsequent dependency transformation have been replaced by the application of a custom Maximum Spanning Tree (MST; Chu and Liu, 1965; Edmonds, 1967) algorithm. This algorithm maximizes the likelihood of edges between the NEs found in the recipe and thus estimates the most likely flow graph. Applying the resulting parsing framework, however, currently leads to a parsing accuracy of around 50%, with most mistakes being caused by the MST algorithm. Later research by Yamakata et al. (2020) also ported the approach to English recipes, with some small modifications due to the flow graph representation not being completely language independent (Mori, Maeta, Yamakata, & Sasada, 2014; Yamakata et al., 2020). Parsing results were comparable to those of the Japanese recipes. Furthermore, these recipe flow graph corpora also inspired some other recipe parsing approaches that led to simplified versions of the presented recipe flow graphs. Simplified graphs were used to make gauging recipe similarity quicker and easier (Donatelli et al., 2021; Yamakata et al., 2013, 2016). However, these simplifications should not lead to deeper recipe understanding or better execution as many details about ingredients and tools are omitted in them.

Dynamics Simulation

As an alternative to directly building graph representations based on recipe texts, some researchers proposed a complementary world-centric modeling approach in which the recipe is understood by tracking state changes in simulation. In their vision paper, Malmaud et al. (2014) suggested that maintaining a latent context during parsing would support correctly interpreting recipe instructions. This latent context represents how the kitchen state would be evolving during recipe execution. This contextual information is needed as the meaning of each individual recipe instruction at the time of processing depends on the availability of resources in the physical environment, the current state of all resources and commonsense knowledge about them. To capture the needed context, Malmaud et al. (2014) propose using a Hidden Markov Model (HMM) in which a recipe sentence and the current state is used to estimate which action on which kitchen entities is being described. They assumed to have access to a cooking simulator that could implement the world dynamics model of the underlying Markov Decision Process, i.e., a simulator that would specify $p(S_t|S_{t-1}, A_t)$, with S_t and A_t respectively being the state

and the executed action at time step t. To further enhance their HMM visual data, in the form of how-to videos or instructional images, could be used to get more reliable estimations. Although the general idea seems compelling, the vision paper's proposed methods currently has not led to an actual implementation.

However, an implementation of a related idea does exist. Bosselut et al. (2018) developed a model called a Neural Process Network in which action and entity relations are integrated into a neural network architecture. They model understanding a recipe as finding a sequence of kitchen state changes induced by executing actions on kitchen entities in simulation. This simulation is performed by updating and tracking a set of a priori specified actions and entity embeddings that encode information along six relevant dimensions, namely location, cookedness, temperature, composition, shape and cleanliness. More specifically, their proposed parsing architecture is composed of five modules. A first module receives one recipe sentence as an input and uses a bidirectional Gated Recurrent Unit (GRU; Cho et al., 2014) to encode every word in that sentence to finally output these encodings as a vector. Using this encoded vector as input a second and third module will then respectively select actions and entities based on their attention distributions. To solve possible issues with sentences in which entities are not explicitly mentioned, the attention distributions from both the current sentence and the previous sentence are considered. The embeddings of selected actions and entities are then passed to the fourth module. This module simulates executing the selected actions on the selected entities and updates the resulting entity embeddings accordingly. The fifth and final module will then use multi-class classifiers to translate the entity embedding updates to discrete changes in the entity states along each of the six previously mentioned dimensions, leading to an interpretable state representation. Experiments with the proposed architecture led to an accuracy of 55% for state changes, which indicate that world dynamics are effectively learned. However, achieving this accuracy already required 65,000 annotated recipes and further improvements will probably require further resource-intensive annotation work.

Multimodal Approaches

People sometimes combine both language and perceptual information in an effort to better understand recipe instructions, as evidenced by the existence of recipes accompanied by images and how-to videos (Malmaud et al., 2015). This insight has led to some recent research focusing on adopting a similar approach for artificial agents by providing multimodal information when parsing recipes. Pan et al. (2020a) proposed a neural encoder-decoder model that fuses image and text embeddings per recipe step to then use these for predicting causal relations between the different steps. By finding these causal relations the model constructs a so-called cooking workflow, which is a DAG with recipe steps as its nodes. Although the level of detail in this cooking workflow representation is insufficient for recipe understanding in the context of execution, it does show that an agent can learn the temporality inherent to a recipe. This means an agent can learn whether two cooking steps can be executed in parallel or only sequentially and what implicit causal effects certain cooking steps might have. Furthermore, the research of Pan

et al. (2020a) demonstrate the feasibility of successfully combining both modalities as performance increased compared to a text-only approach. However, their comparative experiments with text-only and image-only models also indicate that textual information still seems to be the most important modality.

Even more recently, Papadopoulos et al. (2022) proposed a multimodal approach in which a recipe text and an accompanying image of the prepared dish are used to generate a program representation of a recipe. This representation is composed of a sequence of cooking functions. Each function produces an output based on one or more inputs, which could be base ingredients, previous outputs, tools or action modifiers such as a temperature or time specification. This is somewhat similar to the MILK representation (Tasse & Smith, 2008) as the program representation also forms an implicit graph in which functions and parameters are nodes and edges are defined through parameter sharing. Such a structured representation should capture cooking semantics and relational dependencies sufficiently for actual execution by an agent, since enabling execution would then be comprised of providing a usable implementation for each function present in the cooking program. Which functions and parameters are possible in the representation has been established beforehand in a fixed vocabulary. This vocabulary was built using recipes from the public Recipe1M database (Salvador et al., 2017). Papadopoulos et al. (2022) first extracted features for each encountered action or parameter using Sentence-BERT (Reimers & Gurevych, 2019) and then merged semantically identical ones during a clustering phase. Having this vocabulary available, they then proposed a neural encoder-decoder model that can generate a recipe program based on a joint embedding of visual and textual information. Image encoding was based on the Vision Transformer (Dosovitskiy et al., 2020) to obtain visual embeddings and text encoding was based on a similar word-based Transformer architecture to obtain textual embeddings. A program decoder can then use these embeddings to predict a program sequence. Experiments using this architecture and a custom database of more than 3,500 recipes with images and program representations show promising results based on the graph edit distance (GED; Sanfeliu and Fu, 1983) between the predicted program and a golden standard solution. However, it should be noted that computing GED is costly and was therefore computed on only a small subset of test samples.

2.1.4 Robotic Recipe Execution

Real-world robotic cooking requires more capabilities than being able to parse a recipe in natural language to a machine-readable and -interpretable format. Many possibly independent research efforts in the domain of knowledge representation, reasoning, computer vision and robotic manipulation have to be combined into an integrated approach. Although achieving full robotic execution is out-of-scope for this thesis, this can still be seen as our end goal for recipe understanding. Therefore, we will briefly summarize two research studies that demonstrate the feasibility of an agent to physically cook a dish starting from just a recipe written in plain English. It should be noted that these studies have been performed some time ago, so they do not make use of the most recent developments in certain subfields. Therefore, we will mostly focus on providing conceptual

insight into what would be needed to make robotic cooking viable without delving into the actual technologies that were used.

A first extensive feasibility study was performed by Beetz et al. (2011). They experimented with two autonomous robots, TUM-Rosie and TUM-James, that collaborated to bake pancakes with one robot being responsible for fetching everything and the other robot performing actual cooking operations. Their experiments involved five large domains, namely interpretation of procedural text, information usage from knowledge bases, perceptual grounding, robotic manipulation motions and internal reasoning processes. To execute a recipe a robot should first be able to map recipe instructions to a combination of basic operations that a robot already knows how to execute, e.g., picking up an object or putting it down. These basic operations were installed a priori. The mapping approach then used NLP techniques to obtain concepts and combined this with information related to these concepts from so-called ontologies and knowledge bases to eventually come up with a robot action plan. This action plan is declarative in nature, specifying which entities were referred to and which sequential goals should be achieved. Symbolic reasoning based on these goals, a pre-installed map of its environment, information coming from ontologies and knowledge about its own capabilities then leads to a refined plan specifying the procedural actions to be executed. With this refined plan, the robots are then able to perform the required actions to localize and manipulate objects while being guided by perceptual input from their sensors in order to physically make the pancakes. Afterwards, post-execution question answering is possible by having continuously logged data coming from sensors, belief states and the task tree. This question answering demonstrates that an agent can not only execute actions but also understand and explain why he performed them. Additionally, Beetz et al. (2011) also stated that including some form of physics-based simulation could further enhance reasoning capabilities. Moreover, mentally executing actions before physically executing them could help in predicting consequences and thus in avoiding future problems. Lastly, it should be noted that these experiments show feasibility of robotic cooking, but hand-coding of robot actions and providing human assistance to the robots was still required for successful cooking of this one particular dish.

Bollini et al. (2013) also saw the potential benefits of investigating whether a robot cook could be created. They developed BakeBot which is a robot that can bake cookies by reading a list of recipe instructions. Some simplifying assumptions were made, with the most important one being that the kitchen is *mise en place*. This means bowls, small tools and pre-measured ingredients are placed in fixed positions on a work table. Furthermore, similarly to TUM-Rosie and TUM-James (Beetz et al., 2011) the location of larger appliances, such as an oven, are known through a pre-installed model of the environment. The research of Bollini et al. (2013) and Beetz et al. (2011) also have some other ideas in common. According to Bollini et al. (2013) a robot will also have to map recipe instruction to a combination of primitive operations that a robot already knows how to execute, although the primitive operations of BakeBot are more high-level such as pouring or mixing. In the case of BakeBot, this mapping is primarily found through forward search in a simulator. This simulator is based on a predefined state-action space

for the kitchen. To allow finding a good mapping, a probabilistic reward function was also learned from a dataset of recipes annotated with the correct sequence of states and actions. During parsing of recipe instructions, BakeBot then uses this simulator and reward function in order to find the most rewarding sequence of actions. Once this sequence of primitive operations is known, BakeBot executes them one at a time with each high-level operation first being refined into more low-level executable motion operations. Just like TUM-Rosie and TUM-James (Beetz et al., 2011), both perceptual and physical sensors are then used to guide the robot towards physically cooking the actual dish. To conclude, a remark similar to the one for Beetz et al. (2011)'s research should be made for BakeBot. Two real-world baking experiments were performed that show the feasibility of robotic cooking, but the robots still needed human assistance for some actions. Furthermore, the interaction between many different domains and subsystems can lead to problems with robustness. A problem in NLU, perception or robot control can quickly lead to overall failure in dish preparation. Improving all these domains separately should make a future integrated approach more feasible and is therefore worth investing in.

2.2 Benchmarking

When the field of AI first started to really form in the early 1950s and 1960s, the broader goal was to create agents that are capable of simulating human-like intelligence (Haenlein & Kaplan, 2019). During those days progress towards this goal was often assessed by asking somewhat philosophical questions on what it meant to be intelligent, with AI methods being seen as a model of some general cognitive function. The rise of machine learning and especially deep learning, however, has led to a shift in measuring progress towards evaluation of model performances on more application-based objectives (Raji et al., 2021). Since then, benchmarks have taken a more important central position in AI with obtained scores on standard benchmarks often deciding which models are the current state-of-the-art. Clear examples of this can be seen in computer vision and NLP where achieved scores on ImageNet (Russakovsky et al., 2015), GLUE (Wang et al., 2018) and their derivatives have been cited as proof of a model's utility from an academic or commercial standpoint (Raji et al., 2021).

2.2.1 Definition

To ensure a common understanding of what a benchmark is, we will first describe what a benchmark is actually composed of in the context of this thesis. This definition is based on the early work of Jelinek in the 1980s who devised a Common Task Framework to quantitatively assess success on computational linguistic tasks (Donoho, 2017) and more recent views by standardization organizations such as the Standard Performance Evaluation Corporation (SPEC), the National Institute of Standards and Technology (NIST), the Transaction Processing Performance Council (TPC) and others (Bowman & Dahl, 2021; Raji et al., 2021; von Kistowski et al., 2015).

A benchmark consists of three major parts. First, each benchmark has a concrete task or set of tasks for which model performance can be measured and compared. These concrete tasks are generally a stand-in problem for some more abstract capability that we want a model to demonstrate using specified input and output pairs. Secondly, a benchmark provides a dataset or a set of datasets that contains at least test data. Training data is often included but is not required. This data consists of manually, semi-automatically or automatically generated samples that are representative for the aforementioned tasks. The data is generally made at least partially available to users, although submissions on privately held unknown test data are possible too. Thirdly, benchmarks specify a way of evaluating a model's performance on the given tasks by summarizing success and failure of the model on test data using a single metric or a small set of metrics. These metric scores are generally a single number, allowing quick comparisons between different approaches. Finally, the community of users that adopt the benchmark as a shared framework for model comparison is an important aspect as well. Although it is not an immediate part of the benchmark setup, adoption by researchers coming from academia or industry are integral in the eventual utility of the benchmark.

2.2.2 Benefits & Criticisms

The longstanding development and popularity of benchmarks in AI can be explained from the many benefits that come with benchmarking. The existence of standard benchmarks, accepted by a certain research community, allows researchers to directly and empirically compare results. At the same time, it also allows easier interpretation of the results, since properties of the dataset are generally well-studied and better understood among peers (Hennessy & Patterson, 2017; Wagstaff, 2012). Moreover, the recognition researchers receive when improving on state-of-the-art results encourages and motivates them to focus on specific subfields in AI. Benchmarks can thus not only measure progress, but in effect also steer progress by shifting the focus to particular tasks inside a domain (Barbosa-Silva et al., 2022). This is visible historically, with the fields where AI has made the most progress throughout the years being those in which benchmarking has been systematically applied (Donoho, 2017). This progress should not only be explained from researchers' drive to more recognition, but also from the fact that data curation and model evaluation can be a resource-intensive practice in terms of time and monetary cost (Bowman & Dahl, 2021; Kandel et al., 2012). Communal sharing of such resources through benchmarks, therefore, potentially allows speeding up individual research by lowering the costs related to it (Koch et al., 2021). Despite these advantages, issues with well-known benchmarks and general criticisms about benchmarking practices have been brought to light recently.

A first major criticism revolves around the increased centralization of benchmark creation. Koch et al. (2021) analyzed benchmark trends and discovered that widely used datasets generally originated from a small set of high-profile organizations. Although this is not intrinsically problematic, it might lead to a small number of elite institutions consciously or subconsciously shaping general values and the research agenda in the field.

Audits from popular datasets have revealed for example that well-known datasets have a tendency of being biased towards Western countries and white male populations, both in the field of computer vision (Buolamwini & Gebru, 2018; Prabhu & Birhane, 2021; Shankar et al., 2017) and NLP (Dixon et al., 2018; Zhao et al., 2018). Furthermore, the fact that most datasets that are currently provided in popular standard benchmarks are very large-scale (e.g., ImageNet (Russakovsky et al., 2015) and GLUE (Wang et al., 2018)) promote the use of resource-intensive models, requiring resources that might only be available in larger institutions (Dotan & Milli, 2020).

A second criticism has to do with the increased focus of research on a small number of standard benchmarks. Although increased standardization is beneficial for measuring progress in the field, there are some possible problems related to an extensive focus on too few standard benchmarks. First, using a variety of benchmarks could mitigate the propagation of biases that exist in well-known benchmark datasets (Hutchinson et al., 2022). Secondly, benchmark overfitting has been shown to occur causing performance on standard benchmarks not always translating well to challenging real-world scenarios. This can be due to models learning so-called 'shortcuts' and trends present in these datasets (D'Amour et al., 2022; Geirhos et al., 2020) and an over-reliance on specific benchmark metrics instead of other forms of in-depth quantitative or qualitative analysis (Hutchinson et al., 2022; Koch et al., 2021; Kümmerer et al., 2018). Meanwhile, models that might not maximize scores on these metrics could have other beneficial properties related to interpretability, adaptability or resource requirements (Barbosa-Silva et al., 2022; Kiela et al., 2021). Thirdly, despite the large scale and intended diversity of recent standard benchmarks, quick saturation or near-saturation of benchmarks is prevalent (Barbosa-Silva et al., 2022; D. Zhang et al., 2021). Models are already achieving super-human results on standard benchmarks such as SQuAD (Rajpurkar et al., 2016), GLUE (Wang et al., 2018) and extensions made to combat this problem (Rajpurkar et al., 2018; Wang et al., 2019). Such saturation can cause progress measuring to be distorted as increasingly small improvements on these benchmarks might lack significance, either from a statistical or a real-world utility perspective (Barbosa-Silva et al., 2022; Hutchinson et al., 2022; Wagstaff, 2012). At the same time, however, many lesser known benchmarks remain underused as they fail to find more widespread adoption within a community (Barbosa-Silva et al., 2022).

2.2.3 Properties of a Good Benchmark

Benchmark design generally involves multiple, potentially conflicting requirements and considerations which forces trade-offs (Huppler, 2009). Therefore, having multiple benchmarks by different designers is always useful to the field. Nevertheless, there are some properties a good benchmark should always strive to have based on what we know about the high-level composition of a benchmark and what benefits and risks could be related to it. We mostly based these properties on insights that von Kistowski et al. (2015) provided into the development process of benchmarks by standardization organizations SPEC, TPC and NIST. Since these organizations do not focus specifically on AI, we further extended these insights with remarks from different researchers within the field

of AI and NLP in particular.

The following sections will discuss five main properties that a good benchmark should posses. These properties are:

1. Relevance
2. Reproducibility
3. Fairness
4. Verifiability
5. Usability

Relevance

Relevance is probably the most important property for any benchmark. If performing well on the benchmark is of minimal use in the context of domain progress or real-world utility, then communities will have no reason to actually pick it up. Design choices in every component of the benchmark, i.e., the task, the dataset and the metrics, will have an impact on its relevance. Therefore, it is paramount to transparently describe these choices and their impact on the applicability of the benchmark (Huppler, 2009).

The first choice any benchmark designer should make is what the abstract, real-world problem is that the benchmark should help solve. Once this is known, concrete tasks should be developed that are as representative as possible for this abstract task. Despite the recent popularity of benchmarks with broader general-purpose tasks (Russakovsky et al., 2015; Wang et al., 2019), narrower context-specific benchmarks can be more relevant in certain areas (Koch et al., 2021; von Kistowski et al., 2015). In both cases the tasks should be challenging enough, so actual progress can still be made, and the intended breadth of their applicability should be clearly indicated (Huppler, 2009).

Relevance of the task also depends on the relevance of the provided data. Curated data should be diverse enough to allow adequate coverage and transfer of results to real-world use cases. Besides data representativeness, the quality of a dataset is also highly dependent on the quality of the data annotation process. This process should be carefully designed, executed and thoroughly evaluated. This is especially true if data annotation happens via non-expert crowdsourcing, which is common for recent AI benchmarks (Rajpurkar et al., 2016; Russakovsky et al., 2015; Wang et al., 2018). Furthermore, updates and extensions to the used dataset might be needed to ensure benchmark longevity. The real-world problem might evolve over time requiring more recent data or the benchmark might get saturated and thus lose its relevance (Barbosa-Silva et al., 2022).

Finally, using the right metrics is crucial as well. Based on what is most appropriate for the specified tasks, well-known existing metrics could be used or custom alternatives might be a better option. Especially in NLP the use of alternative metrics is not uncommon (Blagec et al., 2020). In such cases, the metrics should be well-documented. Although it is not frequently measured, aspects such as model scalability and resource

need could be interesting to take into account as well from a real-world utility perspective (Ethayarajh & Jurafsky, 2020). The use of an additional diagnostic test set, hand-crafted by experts, could further enhance analysis of a model's strengths and weaknesses (Wang et al., 2018, 2019).

Reproducibility

Reproducibility has always been a cornerstone of the scientific method. Other researchers should be able to independently repeat experiments and get the same result to ensure trustworthiness (Gundersen & Kjensmo, 2018). Similarly, benchmarks that want to accurately contribute to scientific progress should give consistent scores when the same model is evaluated. Trustworthy comparison of methods is impossible if run-to-run arbitrariness in scores would exist (Huppler, 2009; von Kistowski et al., 2015).

In case of benchmarks with leaderboards, independent reproduction of findings by others can be further facilitated by allowing or even requiring researchers to upload an accompanying paper or text describing the model they used to achieve their score (Wang et al., 2019). Such a benchmark feature can even promote benchmark usage as it further gives credit and enhances visibility of researchers' work.

Fairness

Although unfairness towards certain people as a consequence of under- or misrepresentation in popular datasets is definitely an important topic in benchmark design nowadays (Paullada et al., 2021), mitigating that type of unfairness or promoting unbiasedness is more a part of ensuring relevance in benchmark design. Benchmark fairness should be interpreted rather as providing an environment that allows fair competition between benchmark participants. This means the method of evaluation should be clear, well-described and also portable so anyone can evaluate their models based on their own merits (Blagec et al., 2022; Huppler, 2009). Some constraints will generally be put on evaluations and submissions, but these should be well-motivated. The reasons for such restrictions can be either technical in nature or measures to avoid taking advantage of the simplified nature of concrete tasks and subsequent result pollution (von Kistowski et al., 2015).

When it comes to benchmarks with leaderboards, there are two common choices made in this regard. First, unannotated test data is released on which participants should run their models and then submit results for. Benchmark organizers then assess the performance themselves in an automated way. This ensures that no restrictions are placed on the type of used methods and avoids that benchmark organizers have to run submitted models on test data themselves. The latter could be technically infeasible for larger test sets (Russakovsky et al., 2015), even though it might prevent some abuse by having prior knowledge about the test data. Secondly, the number of submissions participants can make in a certain timespan is generally limited. This should help prevent participants that try to overfit their models to the test data (Wang et al., 2019).

On a final note, benchmarks that are designed through committee consensus are often perceived as fairer since benchmark design always requires trade-offs. Such a design approach might not always be efficient or even feasible, but it does make sure that compromises are made in a way that multiple parties agree on its fairness (von Kistowski et al., 2015).

Verifiability

Verifiability corresponds to providing a high degree of confidence that benchmark results are accurate. A frequent choice made in this context is that part of the test data is kept privately, i.e., at least the annotations. Participants then do not assess and publish performance themselves, but a system of automatic validation is responsible for this.

From an academic perspective, verifiability is also closely related to reproducibility of the results. Verifiability could be improved by giving or requiring more details on submission than needed, e.g., showing model insights from a diagnostic test set or requiring an accompanying paper (von Kistowski et al., 2015). Other researchers can then more easily reproduce and verify the results, which simultaneously ensures participants followed benchmark rules.

Usability

Usability relates to a benchmark's ease of use. Both from an economical investment perspective as well as a time and effort perspective. From an economical perspective, the cost of running a benchmark should not be excessive. Although large dataset sizes might be warranted in some cases, the trade-off with the associated expenses of being able to access, train and evaluate on it should be considered as well (Huppler, 2009).

From a time and effort perspective, ease of use can be increased by providing good benchmark discoverability, quick access, adequate documentation on its use and perhaps software tools that support the development or evaluation of models on benchmark tasks (von Kistowski et al., 2015; Wang et al., 2019).

Finally, open and permissive licensing can facilitate reuse, repurposing and extensions of existing benchmarks (Conneau & Kiela, 2018; Wang et al., 2019; Williams et al., 2018), although this is generally more aimed towards enhancing affordability for benchmark designers rather than benchmark users.

2.3 Recipe Corpora

The popularity of recipe-related research and benchmarking in AI has led to the creation of multiple public databases, competitions and benchmark datasets in this domain (Batra et al., 2020; Chang et al., 2018; Gaillard et al., 2017; Salvador et al., 2017). Most of these benchmarks, however, have been created to solve abstract tasks that are not directly related to recipe semantics in the context of execution. A summarizing overview of other recipe-related benchmarks for a variety of tasks can be found in Appendix A, but in this section we will focus the discussion on three publicly available unimodal recipe

benchmarks that do focus on understanding recipes in the context of execution. These benchmarks are the Carnegie Mellon University Recipe Database (CURD), the Recipe Flow Graph (r-FG) corpus and the Recipe Instruction Semantics Corpus (RISeC).

2.3.1 Carnegie Mellon University Recipe Database

In the context of the SOUR CREAM project (System to Organize and Understand Recipes, Capacitating Relatively Exciting Applications Meanwhile), Tasse and Smith (2008) developed both the concise first-order logic based MILK language and the CURD database composed of 260 recipes translated into this MILK language, of which a sample annotation is given in Figure 2.2.

```
<line>
    <originaltext>
        Pour onto a large buttered cookie sheet.
    </originaltext>
    <annotation>
        create_tool(t1, "large cookie sheet")
    </annotation>
</line>
<line>
    <originaltext>
        Pour onto a large buttered cookie sheet.
    </originaltext>
    <annotation>
        set(t1, "butter cookie sheet")
    </annotation>
</line>
<line>
    <originaltext>
        Pour onto a large buttered cookie sheet.
    </originaltext>
    <annotation>
        remove(ing9, t0)
    </annotation>
</line>
<line>
    <originaltext>
        Pour onto a large buttered cookie sheet.
    </originaltext>
    <annotation>
        put(ing9, t1)
    </annotation>
</line>
```

Figure 2.2: A sample taken from the 'Almond Buttercrunch' recipe in CURD (Tasse & Smith, 2008) showing the MILK annotations for "Pour onto a large buttered cookie sheet."

Minimal Instruction Language for the Kitchen

MILK is not intended to be able to identify every detailed action that occurs during recipe execution, but it does provide a way of expressing the temporal step order and the relationships between steps through predicate argument sharing. Recipes should then be interpreted as a sequence of world states describing the introduction of ingredients, their usage and eventual serving as a final dish. Differences between subsequent world states occur implicitly as the result of applying a specified sequence of actions. Parsing recipes thus consists of translating natural language sentences to such a sequence of machine-readable MILK predicates.

To form these machine-readable representations MILK defines three primitive types and twelve actions. The primitive types are *ingredient*, *tool* and *string*. The former two describe entities that can be created or deleted on introduction or serving, while the latter is meant to provide a way to describe human-readable comments and additional information. All of them can be used in the aforementioned action predicates that correspond to void functions with side effects changing the state of the world. Such a world state can be described as a tuple $< I, T, S, I_d, T_d, C >$, with I the set of existing ingredients, T the set of existing tools, S the set of used strings, I_d and T_d specifying which string relates to which ingredient or tool respectively and C specifying which tools contain which ingredients.

Possible actions influencing these states are then *create_ing* and *create_tool* for creation, *combine* and *separate* for combining and separating ingredients, *put* and *remove* for adding or removing an ingredient in a tool, *cut*, *mix* and *cook* for modifying an ingredient in some way, *set* for modifying a tool in some way, *leave* and *chefcheck* to represent not using an ingredient for a specified time or until a condition is fulfilled, *serve* to mark the deletion of an ingredient at the end of the recipe when serving the dish and lastly an open-ended action *do* that can be used to describe any action that cannot be expressed using the other actions.

CURD Composition

The aim of the SOUR CREAM project is to support the development of new techniques for semantic parsing based on cooking recipes. CURD was created in this context and provides a publicly available annotated dataset for this task.

Although the used MILK annotations themselves have been thoroughly documented in an accompanying technical report, descriptions of the data collection and annotation process are lacking. No clear information about the data source is given and Tasse and Smith (2008) only stated that annotations were made by 18 annotators using a custom Java-based GUI tool without specifying annotator experience or inter-annotator agreement. Moreover, the technical report indicates there should be 300 annotated recipes and 350 unannotated recipes in the database while CURD actually only contains 260 annotated recipes. No further information about this discrepancy could be found. The technical report does mention, however, that the recipes currently available in CURD all start with a letter from the first half of the alphabet and further work is thus needed

to ensure the dataset is a representative random sample.

Tasse and Smith (2008) also did not explicitly propose any metrics for evaluating performance on their dataset nor did they make any prior train-test splits, automatic evaluation tools or a leaderboard. A specialized website[2] has been created, but it only provides the dataset itself and the accompanying paper. However, Tasse and Smith (2008) did perform some preliminary parsing experiments themselves in which they used the ratio of correctly identified sentences and the ratio of completely correct instruction series as accuracy-based metrics

Community Adoption

CURD has been an inspiration for other recipe-related benchmark datasets, such as the SIMMR database aimed at the task of recipe comparisons (Jermsurawong & Habash, 2015). However, it currently has not achieved widespread adoption within an NLP community. This could perhaps be explained by the fact that it does not sufficiently meet most of the benchmark properties that we presented in section 2.2. As noted by Tasse and Smith (2008) themselves in their paper, further extensions are needed and these extensions have not yet been made.

2.3.2 Recipe Flow Graph Corpus

After prior research into a machine learning approach for parsing recipe texts into flow graphs required creating their own corpus (Mori et al., 2012), Mori, Maeta, Yamakata, and Sasada (2014) acknowledged the lack of publicly available corpora aimed at recipe understanding. Therefore, they created two corpora consisting of respectively 266 Japanese recipes and 300 English recipes annotated with a flow graph representation in a spreadsheet format. A sample from the English corpus is given in Figure 2.3.

Recipe Flow Graphs

Recipe Flow graphs are computationally tractable, rooted DAG representations that provide an abstraction of linguistic expressions which only preserves information that is relevant for correctly executing steps towards creating the final dish (Yamakata et al., 2020). Nodes of the DAG are labeled with so-called recipe named entities (r-NEs) that correspond to food, tools, actions, states, durations or quantities. Most of the node labels are similar for the Japanese and English corpus, with the exception of some additions that were needed to account for linguistic or general cooking phenomena in English recipes that are not common in Japanese recipes. An overview of these labels can be found in Table 2.1. Edges of the DAG denote relationships between the aforementioned r-NEs. These relationships can be sequencing, part-of, equality, usage and other types of relationships. An overview of these labels can be found in Table 2.2.

[2]https://www.cs.cmu.edu/~ark/CURD

```
1 1 1 Preheat VV0 Ac-B
1 1 9 the AT O
1 1 13 oven NN1 T-B
1 1 18 to II O
1 1 21 170 MC St-B
1 1 25 C ZZ1 St-I
1 1 27 / CC O
```

(a) *.list file with r-NE tags

```
1 1 13 t 1 1 1
1 1 21 o 1 1 1
```

(b) *.flow file with flow graph edges

Figure 2.3: A sample taken from the 'White Chocolate Chip Cookies' recipe in r-FG (Yamakata et al., 2020) showing how a DAG can be constructed for "Preheat the oven to 170°C." using a list and a flow file. In Figure 2.3a the elements on each line correspond to the number of the instruction, the number of the sentence in the instruction, the character number indicating the start of the word in the sentence, the actual word, a POS tag and an r-NE tag with additional BIO information (Ramshaw & Marcus, 1995). The first three elements on each line also serve as a node ID. In Figure 2.3b the first three elements indicate the origin node ID, the fourth element represents the edge label and the last three elements indicate the destination node ID.

Tag	**Full Tag Name**	**Description**	**English only**
F	Food	An eatable product	No
T	Tool	A cooking tool	No
D	Duration	Duration of cooking	No
Q	Quantity	Quantity of food	No
Ac	Action by chef	A chef's action	No
Ac2	Discontinuous Action	Second part of a chef's action	Yes
Af	Action by food	Action of a food	No
At	Action by tool	A tool's action	Yes
Sf	Food state	A food's state	No
St	Tool state	A tool's state	No

Table 2.1: Recipe named entity (r-NE) tags used in the Recipe Flow Graph Corpora (Mori, Maeta, Yamakata, & Sasada, 2014; Yamakata et al., 2020)

Japanese Label	English Label	Description
subj	Agent	Subject
d-obj	Targ	Direct object
i-obj	Dest	Indirect object
F-comp	F-comp	Food used as a tool, e.g., seasoning
T-comp	T-comp	Tool used in an action
F-eq	F-eq	Identical food
F-part-of	F-part-of	Reference to a part of a food
F-set	F-set	Reference to a combination of foods
T-eq	T-eq	Identical tool
T-part-of	T-part-of	Reference to a part of a tool
A-eq	A-eq	Identical action
V-tm	V-tm	Verb of a clause for timing or conditions
other-mod	other-mod	Other relationships

Table 2.2: Flow graph edge labels used in the Recipe Flow Graph Corpora (Mori, Maeta, Yamakata, & Sasada, 2014; Yamakata et al., 2020)

Corpus Composition

Mori, Maeta, Yamakata, and Sasada (2014) state that their corpus can lead to many potential applications, ranging from training models for word segmentation and NER to building intelligent search engines and automatic recipe generators. The main goal, however, is to be able to perform natural language text analysis that can lead to recipe understanding by obtaining a structured meaning representation.

The r-FG corpora were created to help strive towards these objectives within the research community, but at the time of writing neither corpus is actually publicly available. Dedicated websites have been created for both the Japanese corpus[3] and the English corpus[4], but the Japanese website states that the corpus is still under construction and the English website only provides the corpus on request after an approval process via email in which you specify your name, affiliation and purposes.

The websites, however, do already document how to interpret the annotations, how the data was collected and include some general data statistics. The Japanese dataset is composed of Cookpad[5] recipes, with 200 recipes being randomly selected and 66 recipes additional recipes being for the common Japanese dish *nikujaga*. The English dataset is composed of Allrecipes.com[6] recipes, with 200 recipes being randomly selected and 100 recipes being sampled from different dish categories based on popularity criteria. Similar information has also been mentioned in the accompanying papers (Mori, Maeta, Yamakata, & Sasada, 2014; Yamakata et al., 2020).

Furthermore, the accompanying papers also detail the annotation process that was

[3]http://www.lsta.media.kyoto-u.ac.jp/resource/data/recipe/home-e.html
[4]https://sites.google.com/view/yy-lab/resource/english-recipe-flowgraph
[5]https://cookpad.com

followed. Both corpora first used an automatic NER tagger of which the output was then manually corrected for obtaining better node labels. Edge annotations are added manually in a spreadsheet by non-expert annotators. For the English corpus, it was mentioned that two annotators were used and that a high level of inter-annotator agreement was achieved. Similar information was not provided for the Japanese corpus, but they did use a custom program to automatically check whether a spreadsheet is actually a rooted DAG as a form of annotation evaluation.

No metrics were explicitly proposed by Mori, Maeta, Yamakata, and Sasada (2014) or Yamakata et al. (2020) for evaluating performance on their datasets. However, Yamakata et al. (2020) did perform some initial experiments on their English corpus in which they used precision, recall and F_1 scores to present performance results on node label predictions and on edge label predictions.

Community Adoption

The Japanese r-FG corpus has known some use in the Japanese research community, both directly in the development of recipe parsers (Maeta et al., 2015; Yamakata et al., 2016) as well as indirectly by being repurposed for the development of other multimodal benchmarks (Hashimoto et al., 2014; Nishimura et al., 2020). It should be noted, however, that in all of this research the original creators of the r-FG corpus were involved.

The English r-FG corpus has only been used as a training set for a recipe parser in the context of the development of a new English recipe corpus called Aligned Recipe Actions (ARA; Donatelli et al., 2021). ARA aims to improve the alignment of actions between different recipes to gauge their similarity (Donatelli et al., 2021). It used the r-FG corpus as a training set in order to learn how to extract simplified action graphs without ingredients or tools from recipes. However, only the dependency information that is present in the r-FG corpus was used while the rest of the annotation information was ignored.

It currently does not seem like either of the corpora have achieved widespread adoption within an NLP community, especially not in an English NLP community. The dataset itself is well-documented, but it is not so easily accessible. Additionally, many pages on the website for the English r-FG corpus are currently still only available in Japanese. Furthermore, the goal of the corpora mainly seems to be oriented towards providing a development dataset for parsers as no metrics or evaluation tools are given.

2.3.3 Recipe Instruction Semantics Corpus

Jiang et al. (2020)'s aim is to provide a corpus that allows recipe parsing to achieve a basic understanding of its instructions, while limiting the need for domain-specific assumptions in the form of using predefined robot instruction predicates as in CURD (Tasse & Smith, 2008). To support this goal, they created their own annotated corpus of 260 recipes annotated with recipe entity and relationship labels. A sample of this corpus is given in Figure 2.4.

```
T15 AC 297 301 Pour
T16 TOOL 307 336 a large buttered cookie sheet
R10 Arg3_GOL Arg1:T15 Arg2:T16
R11 Arg1_PPT_X Arg1:T15 Arg2:T11
```

Figure 2.4: A sample taken from the 'Almond Buttercrunch' recipe in RISeC (Jiang et al., 2020) showing the entity and relationship annotations for "Pour onto a large buttered cookie sheet.". The elements on each line starting with T correspond to the entity ID, entity type, recipe start position of the expression, recipe end position of the expression and the actual word sequence. The elements on each line starting with R correspond to a relation ID, relation type and entities involved in the relation following PropBank conventions (Kingsbury & Palmer, 2002).

RISeC Representation Structure

To ensure generality of the used semantic understanding principles, RISeC graphs follow the methodology of PropBank (Kingsbury & Palmer, 2002) to assign semantic roles to recipe-related entities. This methodology is based on verb frames, in which a verb specifies an action and relations originating from this verb link it to arguments or modifiers. The relation types are adopted from PropBank, with some extensions where needed, while the entity types are custom recipe-related entities to represent an *Action*, *Food*, *Tool*, *Duration*, *Temperature*, *Condition*, *Purpose* or *Other* for anything else. Furthermore, an additional *Zero Anaphora Verb* label can be added to a verb frame indicating an implicit reference to an earlier concept.

RISeC Composition

Jiang et al. (2020) state that semantic comprehension of recipes can be seen as the combination of three NLP subtasks, namely entity recognition, relation extraction and zero anaphora resolution. RISeC was created to aid in furthering progress on these three tasks by providing a publicly available annotated dataset of 260 recipes.

This dataset can be accessed via a GitHub repository[7] and has a predefined split into a training, development and test set containing a random selection of respectively 50%, 20% and 30% of the samples. This split is directly taken from the SIMMR dataset (Jermsurawong & Habash, 2015), which served as the data source for RISeC. Since SIMMR in turn uses the same recipes as CURD, RISeC actually reuses CURD recipes with different annotations and a predefined split. This means the representativeness issues mentioned for CURD are likely also present in RISeC. Moreover, information about the annotation process is also sparse with Jiang et al. (2020) only stating that three expert annotators were involved and they were in close communication with each other.

Although a train-test split was proposed, no additional evaluation tools are made

[7] https://github.com/YiweiJiang2015/RISeC

available. However, Jiang et al. (2020) did perform some baseline experiments that evaluated identification performance for each entity and relation type separately via precision, recall and F_1 scores. They also evaluated performance on each of the three aforementioned subtasks via micro-F_1 scores.

Community Adoption of RISeC

RISeC might still be relatively recent, but at this moment it has not achieved adoption within an NLP community. It has only been used as inspiration for another recipe-related dataset called Cookdial (Jiang et al., 2022) which aims to improve research on task-oriented dial systems and has been developed by the original creators of RISeC. The lack of adoption could perhaps be explained by the fact that it is based on the same data as CURD, which stated to have some representativeness problems, and the fact that Jiang et al. (2020) state that RISeC's structured representation might need further extensions before being able to solve queries that require explicit reasoning or state tracking.

2.4 Why We Need a New Benchmark

Nearly a decade has passed since Bollini et al. (2013)'s experiments with robotic recipe execution, with research seemingly still not being close to delivering robust robot chefs any time soon. Recipe-related research has definitely continued to know ongoing popularity, but despite the historically shown benefit of speeding up progress there is currently no suitable benchmark for recipe understanding that has been widely adopted. Valid efforts have been made towards providing such a benchmark, but not CURD (Tasse & Smith, 2008) nor the English r-FG corpus (Yamakata et al., 2020) nor RISeC (Jiang et al., 2020) seem to have known much success in the NLP community so far. This is undoubtedly due to a variety of reasons, but certain deficits when it comes to the benchmark properties mentioned in Section 2.2.3 might play a role in this as well.

One of the biggest issues with the currently available corpora is the fact that they only provide an annotated dataset, without specifying how performance on this dataset should be evaluated. Using the right metrics is an intrinsic part of ensuring that benchmark results are both comparable and transferable to the real world, so development of one or more well thought-out metrics seems essential. This would not only increase benchmark relevance, but such a fixed common metric would also improve fairness, verifiability and reproducibility aspects. Even usability might be improved as researchers would not have to devise evaluation methods themselves when performing experiments with such datasets. This is especially true in case automatic evaluation tools would be provided as well, which the existing corpora also lack. Besides the absence of metrics other issues such as inadequate documentation, difficult data access and all annotated data being available as training data will presumably have led to lower adoption rates as well.

Furthermore, Barbosa-Silva et al. (2022) have investigated other attributes that might be correlated to popularity of benchmarks in the domains of NLP and computer

vision. Most of these attributes are either irrelevant for a more context-specific benchmark, e.g., being as general-purpose as possible, or are generally outside of a researcher's immediate control, e.g., having a well-known first author from a top institution. However, Barbosa-Silva et al. (2022) did find that benchmarks with a public leaderboard often have better adoption rates than benchmarks without. On the other hand having competitions related to a benchmark do not seem to have an effect, which might be an indication that a leaderboard's competitive nature is not its main point of attraction. Leaderboard setups also come with well-defined or automatic evaluation methods, which could be an underlying reason for higher adoption rates and thus indeed seems to be an important focuspoint.

In conclusion we can state that even though trade-offs are always required in benchmark design, there are some clear improvements that can still be made when looking at existing recipe benchmarks. Moreover, a lot of advancement is also still possible in the domain of NLU for recipes. For these reasons, we will design and deliver a new benchmark in this thesis that will hopefully be able to measure, steer and speed up progress towards better recipe understanding.

Chapter 3

Defining the Benchmark

3.1 Introduction

This chapter gives an in-depth description of our new benchmark for recipe understanding, which we named MUHAI Recipe Execution Benchmark and have made publicly available online at https://ehai.ai.vub.ac.be/recipe-execution-benchmark. We will provide insight into our general design process and elucidate the reasoning behind made choices. As with any benchmark, trade-offs were needed and in this chapter we will defend the ones that were made by us.

We will first present the three main components of our benchmark, using the benchmark definition given in section 2.2.1. Section 3.2 briefly outlines the concrete stand-in task we are using to approximate the task of recipe understanding in a practical setting. In section 3.3 we will discuss data curation, annotation and general composition of the dataset itself. The evaluation component, which is an important focus in our benchmark, will be discussed in-depth in section 3.4. Although many design choices will have already been explained in the aforementioned sections, section 3.5 will conclude this chapter with a discussion on how our benchmark tries to satisfy the properties of a good benchmark which were mentioned in section 2.2.3. A more detailed analysis of the challenges of tackling our benchmark combined with some specific examples are given later, namely in Chapter 4.

3.2 Task

The aim of the newly created benchmark is to aid in progressing the specific task of understanding recipes that are written in natural language. This is a prerequisite for being able to successfully execute them in a real-world setting and thus for eventually being able to achieve robotic recipe execution as proposed in earlier research (Beetz et al., 2011; Bollini et al., 2013). The concrete task we propose in this context is the parsing of English recipe texts into a special-purpose procedural semantic representation language. This language is based on similar concepts as MILK (Tasse & Smith, 2008),

but is more fine-grained in order to adequately capture all the necessary recipe semantics and express deeper understanding.

Our formal language defines predicates that correspond to executable primitive operations representing meaningful actions within the cooking domain. Arguments of the predicates are logical variables or constants which can represent a variety of cooking-related concepts such as ingredients and tools. Multiple primitive operations can then be connected through sharing of these arguments to form an implicit DAG which represents a complete recipe. We will henceforth call such DAGs semantic networks, of which an example network extract is given in Figure 3.1. If the right semantic network has been obtained through recipe parsing, taking the appropriate actions to cook the intended dish then comes down to executing the resulting network given correct implementations for the used predicates.

3.2.1 MUHAI Cooking Language

Our representation language MUHAI Cooking Language (MCL) provides a way of translating a sequence of natural language sentences into a machine-readable semantic network of executable predicates. Through argument sharing a notion of dependencies between steps is encoded as well in this predicate-argument structure. Such dependencies are derivable from the network by realizing that a shared argument used as input in one predicate is only available once it is provided as output in the other predicate.

Similar to MILK (Tasse & Smith, 2008), recipe execution based on MCL can be interpreted as a sequence of consecutive kitchen states describing the introduction of ingredients, tools and their usage eventually leading to the final dish. However, contrary to MILK, our representation language explicitly includes an input and output kitchen state as arguments in each predicate. The reason for this is twofold. First, including the states makes it clearer that predicate execution leads to a new state of the world and thereby helps in correctly interpreting predicates. It thus improves human readability of the network in addition to machine readability. Secondly, having direct access to input and output kitchen states before performing an action facilitates performing mental simulations, as proposed by Beetz et al. (2011), which could enhance reasoning capabilities and prevent unintended consequences. Backtracking to investigate the effect of alternatives is possible for example.

Arguments

Arguments of primitive operations can be divided into five broad categories covering most kitchen-related concepts and objects. These categories are kitchen states, food, tools, mode specifiers and quantity specifiers. The arguments themselves can either be variables or constants, with the former mostly being used when arguments need to be propagated throughout the semantic network via argument sharing between primitives.

Kitchen States A kitchen state represents the relevant world in the context of cooking a dish. This includes indicating the availability of food preparation areas, appliances,

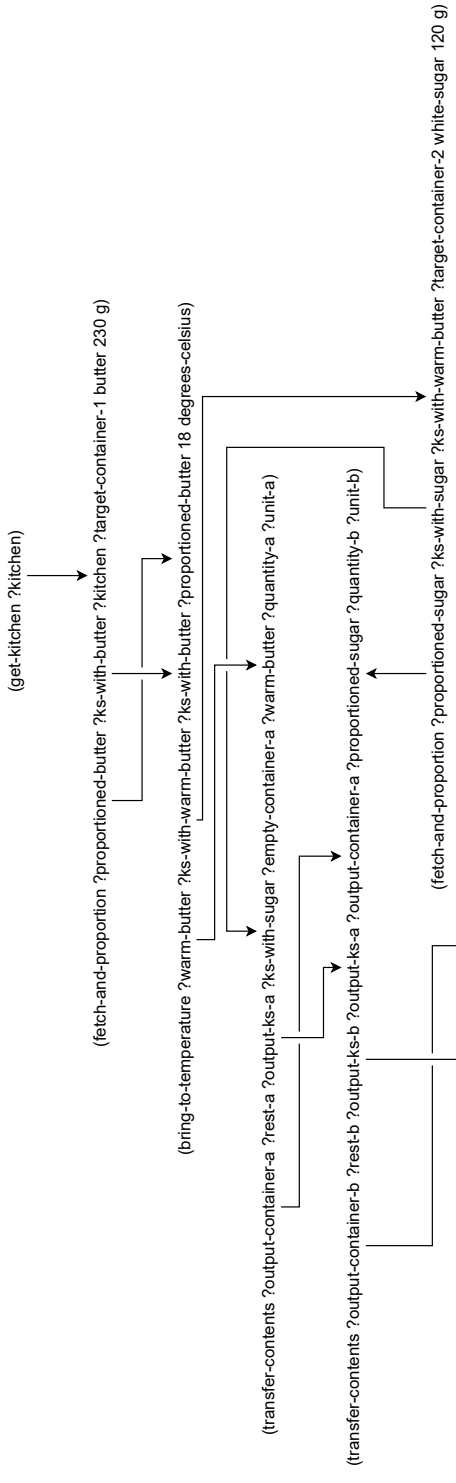

Figure 3.1: Semantic network for beating together some warm butter and white sugar.

cooking utensils, food products and their properties at that point of execution. These properties could concern low-level physics-based characteristics, e.g., the temperature of cookies, or more high-level abstract cooking characteristics, e.g., specifying that cookies are 'baked'. As mentioned before, they are mostly provided to facilitate performing mental simulations and improve execution order readability.

Food Food can denote base ingredients, intermediary food products or the final dish. Furthermore, food arguments can be used to specify ingredients in a conceptual or a referential way. In the conceptual case, we use a food argument to specify that we need some 'butter' for example without indicating which specific instance of 'butter' we should use, e.g., 'grease the pan with some butter'. In the referential case on the other hand we do refer back to a specific instance of 'butter' that should be used, e.g., 'add the proportioned butter'.

Tools Tools can denote cooking utensils as well as cooking appliances and other kitchen commodities that serve an ancillary capacity. Utensils are mostly smaller tools such as a knife or a whisk while appliances are larger machines such as an oven or a fridge. Kitchen commodities cover food preparation and storage areas such as a countertop, kitchen cabinets or pantries.

Mode Specifiers Mode specifiers indicate the manner in which an action should be performed, e.g., in the instruction 'shape the dough in a crescent shape' the indication of 'in a crescent shape' specifies the mode of the shaping action.

Quantity Specifiers Quantity specifiers can be used to provide numerical information that is important for correct execution of a cooking action. This includes specifications of durations, temperatures or weight and volume measurements. Generally such a specification consists of two arguments that should be interpreted together. One argument specifies the actual numerical value while the other argument specifies the unit of measurement that is needed to correctly interpret the magnitude of this numerical value. In the expression 'bake for 10 minutes' both a numerical argument '10' as well as a unit argument 'minute' is needed in order to correctly execute the baking action for the right duration.

Primitives

The predicates in a semantic network correspond to the atomic actions that an artificial agent is able to perform. A level of abstraction was chosen so primitives would specify atomic actions that are closely related to cooking, e.g., a 'bake' operation, instead of more general fine-grained operations such as picking up or putting down objects. We leave these more general operations as implementation-dependent choices, similar to the representation language used in the experiments of Bollini et al. (2013).

Although the chosen implementation of each predicate will in effect determine its practical execution, each primitive does have an intended meaning in MCL. Currently our representation language consists of 38 primitive operations representing common, sufficiently distinct cooking actions. Broad categories of the type of primitives are discussed in the following paragraphs and a detailed explanation of each separate primitive is available in Appendix B and the benchmark's accompanying documentation. Some primitives can in practice be categorized under multiple categories, in which case we mention them in the one that best suits their most prominent behavior.

Kitchen State Loading There is one special-purpose primitive that is expected to be present in every MCL semantic network, which is *get-kitchen*. This primitive is responsible for loading in the initial kitchen state and thus for providing the necessary contextual information for interpreting a recipe. The initial kitchen state could influence the interpretation of recipe texts, because it could alter what is possible and thus what the semantic network should look like to allow recipe execution.

Location Altering There are primitives needed to locate and partially or wholly move ingredients or tools. The primitives *fetch* and *fetch-and-proportion* can be used to find tools or ingredients and bring them to a kitchen work area. The primitives *transfer-contents* and *transfer-items* will transfer food products from one location to another in a less or more careful manner while adhering to a certain placement pattern for the latter primitive.

Food Combination Almost every recipe has operations related to the combination of food products to form other food products. Homogeneous mixtures can be made via the primitives *mix*, *beat* or *shake*, while a more heterogeneous mixture can be created through the use of *mingle*. The first two primitives are mostly used to create dough-like products, while mingling is generally used to create something like salads. Shaking will mostly be used to further combine liquids.

Combining food products in a more layered fashion is supported through the use of other primitives such as *spread*, *sprinkle* and *dip*. These will all use a different mechanism to create an outer layer made from a food product for another food product or potentially even for a tool.

Food Separation Operations related to separating food products to form multiple other food products are also quite common. Our simulator supports two types of separations. The first type effectively splits up a food product into different types of other food products. This is the case for the primitives *sift*, *drain*, *crack*, *separate-eggs*, *peel* and *seed* that can for example be used to split off egg shells, yolks and whites from eggs or peels and seeds from vegetables. The second type splits up a food into multiple smaller versions of the same food product, which is the case for portioning-type primitives such as *portion-and-arrange* and *cut*.

Food Manipulation Many cooking actions change the properties of a food product. These properties could be related to shape, texture, temperature and other physics-based or abstract cooking-related characteristics. Some might require the use of tools, while others just require manual manipulations or waiting. Primitives in this category are *shape*, *mash*, *flatten*, *melt*, *fry*, *boil*, *bake*, *wash*, *bring-to-temperature*, *refrigerate* and *leave-for-time*. The primitive *leave-for-time* is included here because it might have a cooling effect based on differences between the food's temperature and the ambient temperature.

Tool Manipulation Many cooking actions change properties or the general state of a tool. These changes can have an impact on the usability of the tool for certain other actions. MCL supports three types of tool manipulations. The first type concerns changing settings of for example electrical tools, which is done by the primitive *preheat-oven*. The second type concerns addition or removal of other tools to modify the functionality of a tool. Using the primitives *cover*, *uncover* and *line* on a container will change the behavior of other primitives that use that container later on. Lastly, the addition of food products could also modify the functionality of a tool in similar way as the second type. Such additions are possible via the primitives *grease* and *flour*.

3.3 Dataset

The data that will be provided with our benchmark is a test set that is composed of publicly available English recipe texts, accompanied by gold standard solutions. No training data has been provided, which is a deliberate design choice. This choice was made for two main reasons. The first reason was preventing the problem of benchmark overfitting. Completely separating test and training data collection could potentially avoid correlations between both that can be easily discovered by large data-intensive models while being undetectable by humans (Barbu et al., 2019). Only providing a test set could therefore lead to better generalization of the results. Secondly, the absence of a training dataset promotes participation of a larger variety of models. Since we put no limitations on the training data that can be used, a larger variety of both resource-intensive and resource-efficient approaches have a fairer chance at showcasing their unique benefits. If we would have already provided a large-scale labeled training set for example, it generally overly promotes supervised deep learning methods (Dotan & Milli, 2020).

3.3.1 Data Composition

The current test set in our benchmark is composed of thirty recipes from five different online sources of English recipes. These sources are AllRecipes.com[1], SimplyRecipes[2],

[1]https://www.allrecipes.com
[2]https://www.simplyrecipes.com

Food.com[3], The Spruce Eats[4] and Cooks.com[5]. Five different sources were used to ensure a bigger variety in writing style and recipe formats, which could prevent some potential biases that might have been present otherwise. SimplyRecipes for example only publishes recipes after they have been thoroughly reviewed, tested and potentially rewritten by experts while Cooks.com has more recipes written by cooking enthusiasts and laymen.

Although five different sources for thirty recipes seems adequate, thirty recipes in itself cannot possibly generalize to the capability of cooking every possible dish. This design choice, however, seemed inevitable in the short term from a practical standpoint. Significant time and effort had to be put into annotation and the development of evaluation methods if we wanted to ensure a high enough quality. Considering the available resources, a trade-off had to be made between benchmark quality and dataset quantity. Therefore, we eventually chose thirty recipes in which many of the linguistic challenges are present that have been discovered in previous research, which were mentioned in section 2.1.2 and will be further discussed in Chapter 4. Additionally, we chose to have our recipes focus on only two types of dishes, namely salads and baked goods. These subdomains seem distinct enough to require a more general understanding of the cooking process in order to perform well on both. Moreover, if our benchmark would lead to real-world experiments with robotic execution it is also more probable that only a small number of recipes will be focused on at first instead of hundreds, as this was also the case in earlier experiments (Beetz et al., 2011; Bollini et al., 2013).

Nevertheless, it should be noted that our benchmark setup is currently more aimed at surveying the current landscape by providing an opportunity to really analyze existing issues and showcase the benefits of different approaches. Such a setup is definitely useful for progressing the field further, but it cannot be seen as a reliable measurement of how far the field has already progressed globally due to the limited test set size. Further extensions can and probably will be made in the future, however, to cover more dish types and to allow the benchmark to be a more global progress indicator as well.

3.3.2 Data Format

Since all recipes currently originate from online sources, they might be subject to change or even disappear over time. Therefore, we extracted the relevant recipe information from those websites and included it into our benchmark instead of referring to the original source. A metadata file, however, does mention the source and access date for each of the extracted recipes.

Although the presentation format of each recipe might be slightly different in the original source, we chose to use a consistent and structured XML format to represent the recipes in our benchmark. Being able to create models that can extract necessary recipe information from different websites using different representation formats could be interesting research as well, but it is not the focus of this benchmark.

[3]https://www.food.com
[4]https://www.thespruceeats.com
[5]https://www.cooks.com

The structured format used in our benchmark consists of four important components. First, a recipe ID is included which is required to identify the recipe a prediction is for. Secondly, the human-readable recipe title is mentioned which could be useful as contextual information since it often states the type of dish that should be prepared. Thirdly, a recipe file also has a list of all initial ingredients that will be used to prepare the dish. Lastly, each file also contains a list of recipe instructions that should be executed in order to prepare the final dish starting from the initial ingredients. An example recipe file containing all these components is shown in Figure 3.2

```
<recipe>
    <id>easy-banana-bread</id>
    <title>Easy Banana Bread</title>
    <ingredients>
        <ingredient>60 grams butter</ingredient>
        <ingredient>2 eggs</ingredient>
        <ingredient>200 grams sugar</ingredient>
        <ingredient>3 bananas, mashed</ingredient>
        <ingredient>1 tsp. vanilla</ingredient>
        <ingredient>200 grams self-rising flour</ingredient>
    </ingredients>
    <instructions>
        <instruction>
            Cream together butter, eggs and sugar until smooth.
        </instruction>
        <instruction>
            Add bananas and vanilla; beat well.
        </instruction>
        <instruction>
            Mix in flour.
        </instruction>
        <instruction>
            Bake at 165°C for about 1 hour.
        </instruction>
    </instructions>
</recipe>
```

Figure 3.2: The structured representation of the 'Easy Banana Bread' recipe, as it is included in our benchmark. A recipe consists of an id, a title, an ingredients list and an instructions list.

3.3.3 Data Annotation

Annotation Format

As mentioned in section 3.2.1, our recipe annotations are semantic networks composed of predicates connected through shared arguments. These networks are more meaningful as a whole due to common recipe idiosyncrasies such as missing steps, ellipses and contextual references. Therefore, we do not consider separate annotations for individual

instructions but provide one gold standard annotation for the entire recipe.

Furthermore, we do not explicitly separate ingredient lists from instructions either. In practice each ingredient that is mentioned in such a list requires an action, namely fetching and proportioning an instance of that ingredient. Therefore, both the ingredient list and the list of instructions lead to annotations representing procedural cooking actions. More subjective declarative sentences such as "I like it when the dish has marinated a bit longer!" generally do not have an impact on the semantic network.

A final characteristic of our semantic networks is that they are expected to start with a special *(get-kitchen ?initial-kitchen)* primitive that loads in the initial kitchen to start from. This is included for ensuring correct contextual understanding of what an instruction entails given a specific starting situation, as explained in section 3.2.1.

Lastly, it should be noted that for convenience sake our annotations are stored in files as a flat sequence of predicates even though they should still be seen as an implicit network with connections being made through shared arguments. An example of such a flat sequence is given in Figure 3.3

```
(get-kitchen ?kitchen)
(fetch-and-proportion ?proportioned-butter ?ks-with-butter
    ?kitchen ?target-container-1 butter 230 g)
(bring-to-temperature ?warm-butter ?ks-with-warm-butter
    ?ks-with-butter ?proportioned-butter 18 degrees-celsius)
(fetch-and-proportion ?proportioned-sugar ?ks-with-sugar
    ?ks-with-warm-butter ?target-container-2 white-sugar 120 g)
(transfer-contents ?output-container-a ?rest-a ?output-ks-a
    ?ks-with-sugar ?empty-container-a ?warm-butter
    ?quantity-a ?unit-a)
(transfer-contents ?output-container-b ?rest-b ?output-ks-b
    ?output-ks-a ?output-container-a ?proportioned-sugar
    ?quantity-b ?unit-b)
(beat ?beaten-mixture ?ks-with-beaten-mixture
    ?output-ks-b ?output-container-b ?mixing-tool)
```

Figure 3.3: Flat sequence representation of a semantic network in which some warm butter and white sugar are beaten together. The implicit graph representation for this recipe extract is given in Figure 3.1.

Annotation Process

Initial annotations of ten recipes were performed by ten computer science students from Vrije Universiteit Brussel (VUB), including this thesis' author, under the guidance of two university professors from VUB who are experts in the domain of NLP. To ensure high-quality results, group sessions were held in which the correctness of each annotation was discussed in detail.

However, as will be explained in section 3.4, evaluation will mostly be performed through cooking simulations. While building and testing this simulator, it became clear

that modifications to the representation language and thus also to the previously made annotations were required. This new annotation iteration, which still borrowed a lot of insights from the previous iteration, was performed by this thesis' author with the help of his promotor who also coordinated the previous annotation efforts. Furthermore, while following the same process twenty new recipes have been annotated by this thesis' author to extend the dataset.

To verify correctness of the annotation, this final annotator thoroughly tested each recipe in simulation and executed them in the real world as well. This should lower the chance that unmentioned implied steps or ellipses were missed during the annotation process.

3.4 Evaluation

The evaluation component of our new benchmark has been the main focus during design and development, because our background study seemed to suggest it is one of the main factors that is missing in existing benchmarks for recipe understanding. Furthermore, as explained in section 2.2.3, metrics should be chosen in a way that achieved performance on them would maximally mimic real-world performance. Since our benchmark task is ultimately meant to improve recipe understanding in the context of procedural execution, most of our evaluation methods are based on measuring execution success.

Real-world robotic execution for every parsing result would not be feasible, but approximating such a setting with a kitchen simulator is. Therefore, we developed a simulator that can execute semantic networks in our representation language when given an initial kitchen environment. Performance can then be evaluated using special-purpose metrics that have been inspired by metrics previously used to gauge performance on compositional directives with non-reversible state changes (Shridhar et al., 2020).

These simulation-based metrics are 'goal-condition success', 'dish approximation score' and 'recipe execution time'. In case benchmark users would prefer to not use the simulator, we also included the commonly used graph matching algorithm Smatch that directly compares semantic networks. However, the latter graph-based metric does come with some downsides compared to the simulation-based metrics as will be discussed in section 3.4.3.

How to use our simulator and its evaluation tools in practice is explained with examples in Chapter 4 and our benchmark's accompanying documentation. In the rest of this chapter we will focus on general design choices and the reasons for making them.

3.4.1 Simulator

We want a simulator that can be easily used to aid in both model development and evaluation. For these purposes a simulator should fulfill at least four requirements. First, it should provide an implementation for each of the primitives in our representation language. Secondly, it should be able to represent and manipulate the arguments that are propagated from one primitive operation to another. Thirdly, it should be able

to find a contextually valid execution order for all primitives in the network since a semantic network is not necessarily a clear sequential structure. Lastly, it should be portable enough to support most users. A computational system that supports all these prerequisites is Incremental Recruitment Language (IRL; Spranger et al., 2012; Van den Broeck, 2008) which is part of the Babel toolkit for multi-agent experiments on emergent communication (Loetzsch et al., 2008; Nevens et al., 2019).

To fulfill the first requirement IRL provides an interface which can be used to implement cognitive operations. These cognitive operations can be seen as multi-directional predicates that compute a set of output arguments from a set of input arguments, as is normally done in constraint languages. The primitives in our representation language can be seen as a case of such cognitive operations.

The second requirement is satisfied through IRL's support of semantic entities. These semantic entities can be seen as data which can be bound to the arguments of cognitive operations. Semantic entities can be used for any type of data, which means we can use such entities to define kitchen states, modes, quantities and food or tool concepts and instances.

IRL also fulfills the third requirement as it can execute networks of cognitive operations by performing a search process in which operations are chosen for execution by determining what bindings are available at each step. In other words, it can search for a valid execution order of a procedural network in a contextual data-flow driven way.

The computational system of IRL also meets the last requirement as the system is written in Common Lisp and supports most major Lisp implementations on at least the major platforms. More specifically, it has been verified to run for the Lisp implementations CCL, SBCL and LispWorks on Linux, Mac OS and Windows platforms.

In addition to the IRL system we also chose to use the web interface module of the Babel toolkit. This web interface provides interactive visualizations of the network execution process in which model developers could further inspect kitchen entities at each step of execution. This allows for easier in-depth analysis of results and might lead to more insight into the performance of their model and possible remaining issues. An example of a web interface visualization is given in Figure 3.4.

Cooking Primitives & Kitchen Entities

IRL already provides a useful system for implementing and running cooking simulations, but implementation choices for the cooking primitives and semantic entities will still significantly impact simulation behavior. Therefore, in this section we will discuss some of these choices.

A first technical choice is that we implemented all primitives and entities symbolically using our own cooking ontology to further enhance interpretability of the simulation process and facilitate analysis of execution details. A kitchen state will envelop all details of the kitchen such as available ingredients, tools and their locations. A symbolic representation therefore supports investigating in detail what contextual changes occur at each execution step in a human-readable way.

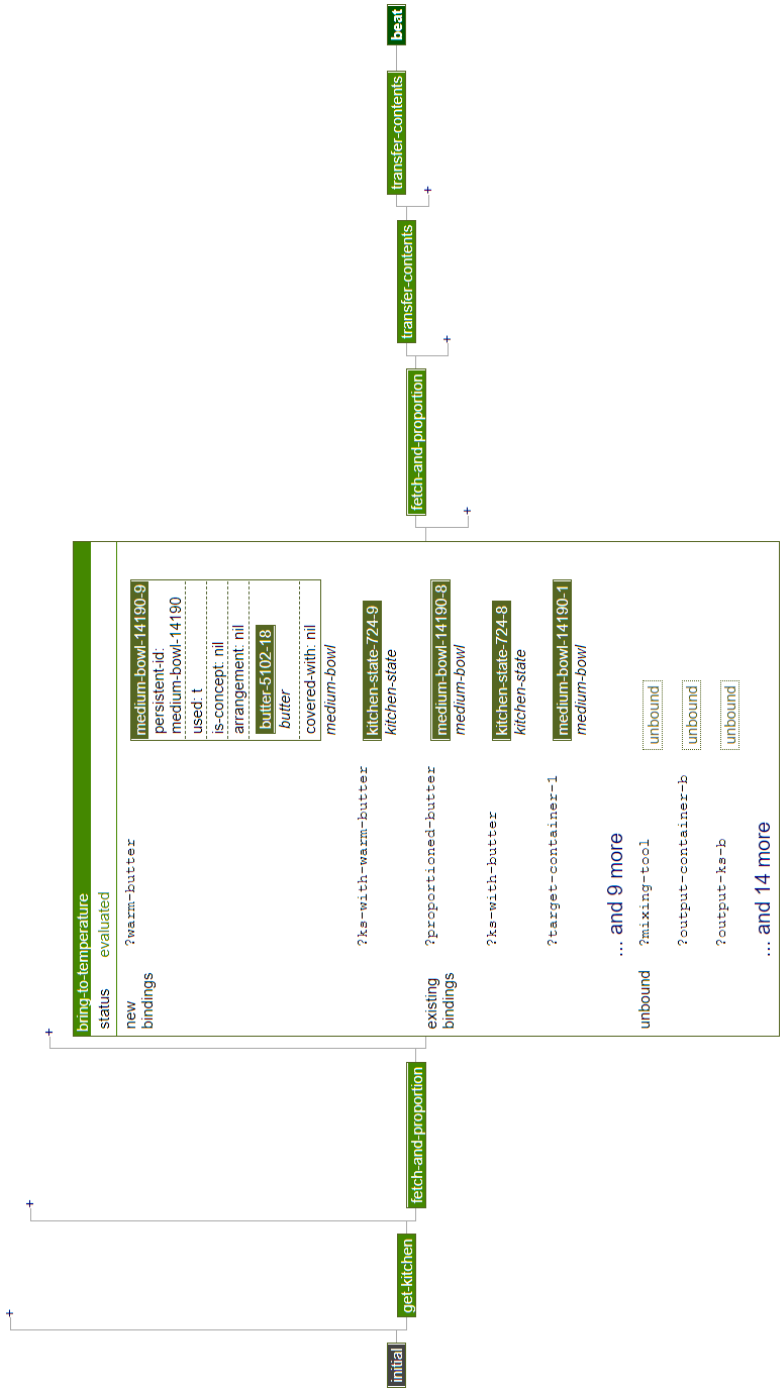

Figure 3.4: Interactive visualization of the simulator's execution process for a small recipe extract in which some warm butter and white sugar are beaten together. All steps and entities can be expanded further for more details. This visualization was obtained by simulating the semantic network given in Figure 3.3.

Secondly, a special type of argument was created and used in our primitives to circumvent a problematic technical characteristic of IRL. The currently available IRL search process will fail and stop execution of a semantic network as soon as no primitive operation can be found for successful application at a certain time in processing. In the case of recipes failure of prior steps, however, do not necessarily lead to failure of future steps. Not finding flour to add to the dough for example has no effect on the chocolate dip that should be created later on. Therefore, a special type of argument was created and used in our primitives that indicates a primitive operation has a 'failed object' as a result while still allowing the primitive operation itself to succeed. This approach ensures a solution will be found in which as many steps as possible are actually executed leading to more reliable evaluation results.

Thirdly, kitchen entities and primitive arguments can be typed in IRL. We used this property to more explicitly model some general world knowledge into our simulator. Typing entities can be used to encode such general world knowledge by organizing types in a hierarchical way in an ontology. Such hierarchies make it explicit for example that both 'white sugar' and 'brown sugar' are a type of sugar and can therefore be used where 'sugar' is needed. To ensure the aforementioned 'failed object' can be used everywhere, this type is put at the absolute bottom of the hierarchy by extending all other types. Using typed arguments also ensures execution simulation more closely approximates real-world scenarios in general. We can for example ensure a network operation to flour a tray with onions will fail immediately by specifying the flouring primitive needs an argument that actually allows flouring.

Other realistic affordances have been included as well, but not necessarily in a technical way. Some conditional checks have been included in primitive implementations that lead to early operational failure if they would do the same in the real-world. Shaking an open jar with liquid for example will fail as the liquid would not remain in the jar. Nevertheless, we should indicate that the simulator is currently not a full-fledged physics-based simulator of the real world. Some common affordances have been included but not all possibilities have been accounted for. The used metrics, however, will still punish mistakes even if the simulation fails later than needed.

Although many affordances have been included, some simplifying elements have been added as well. Firstly, similar to the experiments of Bollini et al. (2013) we assume that the kitchen is *mise en place*. All ingredients are distributed in bowls and primitives will generally handle bowls with ingredients or intermediate products in them instead of directly handling an ingredient itself. This seems to be a reasonable assumption as robotic detection and manipulation of labeled bowls is more likely in the foreseeable future considering existing limitations in the field of computer vision and robotics in general (Mason, 2018). Secondly, to prevent an abundance of fetch and other less cooking-oriented operations some primitive implementations support default values for certain arguments. If no mixing tool has been specified when the primitive for beating some eggs is executed for example, the simulator will automatically fetch a whisk by default. Explicitly including a fetch operation in the semantic network and using its ouput for the beating operation, however, will work as well and will not be punished in

the simulation-based metrics. The documentation provided with the simulator mentions for each primitive which arguments are mandatory and which are optional.

Finally, temporality of instructions is encoded into our simulator by including a time of availability for each argument. Each primitive will specify the time of availability of its output arguments based on the time of availability of its input arguments. This ensures that we can detect temporal dependency relationships between executed operations and thus discover which cooking operations could be executed in parallel and which could not.

3.4.2 Simulation Environments

To be able to evaluate solutions from models, the simulator has built-in simulation environments for each of the thirty recipes included in the test data. Such a simulation environment is comprised of four main components, namely an initial kitchen state, a gold standard semantic network, the final dish that is expected and a recipe execution time.

The initial kitchen state provides the intial context to start from and gets loaded in by the *get-kitchen* operation, which should be present in every semantic network as explained in section 3.3.3. All test recipes currently start from the same initial kitchen state, which contains all the ingredients and tools that can be reasonably used to execute any of the test recipes. A detailed inventory is provided in Appendix C and the documentation accompanying the benchmark.

The gold standard semantic network is our solution for the recipe from the test set. This solution tries to obtain the final dish in the most efficient way. This means tools are reused as much as possible instead of constantly fetching a new tool, which would lead to more required cleaning and lost time. If a recipe states to 'cut the tomato and the cucumber', for example, the same knife will be used for both ingredients in the gold standard solution even though using two different knives would lead to the same outcome. However, recipe steps are also followed as sequentially as possible when order does not explicitly matter. If a sentence in a recipe step states to add ingredient 1 and then ingredient 2, the gold standard semantic network will add these ingredients in this order as well. This could be important to realize when trying to interpret certain metric results.

The expected final dish is stored because the nonsequential nature of networks can complicate automatically determining which argument of a semantic network can be considered the final output. Some recipes might even contain small instructions concerning cleanup or how to handle surplus ingredients, which means the final instruction does not necessarily lead to the final dish. Therefore, we explicitly specify which simulation object is considered the cooked dish.

Recipe execution time is based on the time of availability concept mentioned in section 3.4.1. We store when the last argument would be available if we follow the gold standard semantic network and consider this to be the total execution time for the cooking process in simulation. This is mostly stored to allow comparing the time efficiency of semantic network solutions for this recipe.

3.4.3 Metrics

The main reason our benchmark provides a simulator is to allow evaluating performance using special-purpose simulation-based metrics. Nevertheless, we still included one evaluation metric that uses a semantic network matching process for score computation as it is commonly used in practice. Supporting both familiar and newer special-purpose evaluation tools in our benchmark could be useful to increase adoption, even if we consider the latter to be more informative. This led to four evaluation metrics being included in our benchmark, namely:

- Smatch score
- Goal-condition success
- Dish approximation score
- Recipe execution time

Of these four evaluation metrics the latter three are simulation-based. We prefer these simulation-based metrics over Smatch scores and other metrics for comparing semantic graph structures for two main reasons. First, these non-simulation-based metrics try to determine semantic overlap between semantic networks by computing how much is shared structurally with every node and edge being considered equally valuable (Cai & Knight, 2013; Papadopoulos et al., 2022). However, in the real world certain mistakes while cooking will have a higher impact on the end result than others. A weighted evaluation method could perhaps mitigate this issue, but the second reason concerns a problem that is harder to mitigate without simulation. It is possible that a semantic network contains more actions or a permuted sequence of actions when compared to the gold standard while still leading to the same final dish, albeit in a more inefficient way. A graph matching score therefore does not always give a good indication of actual performance for recipe understanding in the context of execution. Some prior research did attempt to take such effects into account, but noted that high computational cost of graph matching in general already complicates evaluation (Papadopoulos et al., 2022).

The idea behind each of our metrics will be explained in detail in the following sections, with examples in which the scores are actually computed also being available in our benchmark's accompanying documentation.

Smatch Score

The semantic matching tool Smatch is a Python library[6] that has been developed by Cai and Knight (2013) as an efficient evaluation metric for determining the semantic overlap of two semantic feature structures, such as a predicted semantic network and a gold standard semantic network.

[6] https://github.com/snowblink14/smatch

In order to determine this semantic overlap the Smatch tool first converts semantic structures to triples representing a conjunction of logical propositions. These triples can take one of the following three forms:

- ('instance', variable, concept) for triples denoting that a variable represents an instance of a given concept;
- (relation, variable1, variable2) for triples denoting that two variables are connected to each other by the given relation;
- (attribute, variable, value) for triples denoting that a specific attribute of the given variable has a certain constant value.

Once the triples are obtained, the Smatch tool then computes the F-score of the triples in the second network against the triples in the first network as a measurement for the amount of propositional overlap between the two. This F-score can be computed using formula (3.1), with M being the number of matching triples, T being the total number of triples in the first network and G being the total number of triples in the second network.

$$F = 2\frac{\frac{M}{T}\frac{M}{G}}{\frac{M}{T} + \frac{M}{G}} \tag{3.1}$$

However, there are multiple possible values for M since variable names are not necessarily shared between the two semantic networks and different variable mappings are thus possible. Therefore, Smatch computes the maximum F-score that is possible when considering all possible variable mappings. Due to the computational complexity, this maximum F-score is actually approximated through the use of a hill-climbing method, but experiments of Cai and Knight (2013) have shown that this method is both efficient and effective.

Although the hill-climbing method and F-score computation from the Smatch Python library could be used directly for our use case, the library does not support conversion to triples for our type of semantic networks. This conversion step was therefore implemented ourselves by

1. mapping each encountered primitive and variable to an instance;
2. creating an argument relation between each primitive name and its variable arguments;
3. creating an argument attribute for the primitive name with a fixed value every time a constant argument is encountered.

For further clarification, a brief example of such a conversion is given in Appendix E.

Goal-Condition Success

Goal-condition success is a score between 0 and 1 representing the ratio of goal-conditions that have been achieved to those that were minimally needed to finish the task of cooking the complete dish. It is probable that a model did not manage to understand the recipe adequately enough in order to reach a kitchen state in which the final dish is available. Goal-condition success tries to measure how many steps a cooking bot is removed from creating this final dish based on his current understanding of the recipe. Therefore, complete success of the recipe understanding task is only achieved if goal-condition success is 1.

The goal-conditions that are used in each recipe are based on the primitives that are present in the gold standard network, excluding *get-kitchen* as its inclusion is convention and not really an indication of understanding. Each gold standard semantic network is annotated in a way that the network is as minimal and efficient as possible without excluding any important execution information. Therefore, the output variables of each primitive in the gold standard can be seen as a set of minimum goal-conditions that should be reached during the cooking process. The output kitchen state, however, is excluded as a goal-condition. This is needed because 'cut the tomato and the cucumber' for example can lead to a kitchen state with one or two dirty knives, but both states should be considered equal with regards to reaching the goal-condition of having a cut tomato and cucumber on the countertop. Furthermore, the order in which goal-conditions are reached does not matter either. Multiple paths could be taken to reach the final dish and even different combinations of primitives might be possible to reach it. On each of these paths, however, the aforementioned gold standard ouputs are expected to be encountered at some point.

In practice, goal-condition success is computed by

1. collecting all outputs from the gold standard network and the predicted network;
2. checking whether each gold standard output is present in the predicted outputs, with each predicted output only being allowed to match once in case the same goal-condition should be encountered multiple times;
3. storing all reached and unreached goal-conditions to further facilitate model analysis;
4. computing the ratio of reached goal-conditions to total goal-conditions and obtaining a score between 0 and 1.

Dish Approximation Score

The dish approximation score is a score between 0 and 1 and represents an alternative estimation of how far a model was from reaching the final dish. Achieving complete task success or a goal-condition success of 1 seems improbable for most models, but only determining how many goal-conditions were left to be reached does not necessarily

indicate how closely their dish resembles the gold standard final dish. Therefore, we also need a score that approximates how similar two dishes are.

Just like a human would still have some subjective influences when comparing the taste of two dishes, we are aware this score is also based on some subjective choices by us of what is important. However, we tried to base our computation on criteria that seemed sensible as much as possible.

Our dish approximation score is based on a custom point awarding system in which symbolic properties of a predicted dish and the gold standard dish are compared and points are awarded when these properties match. Since it is uncertain which dish is the final dish in a predicted semantic network, scores are computed for each predicted output food product and the maximum obtained score is returned as the final dish approximation score for the network. To be able to distinguish importance certain types of properties have a higher number of points that can be earned than others. Moreover, the dish approximation score is computed as a weighted sum of ratios of earned points to the maximum obtainable points. We will elaborate on the exact computation process in the following paragraphs.

A final dish in our simulator consists not only of the actual food product, but also of the way it is served in a certain container. This container could for example be a specific type of baking tray greased with butter, be placed on a wire rack in the kitchen or contain a specific number of portions of the made food product. All these kind of more presentation-oriented properties are awarded one point each if they match. Since they should have a less significant impact on the actual taste, we compute the ratio for these points separately and give them a factor of only 2% in the final weighted sum. The most important aspect in achieving a similar taste is the presence of correct ingredients in the predicted dish. Therefore, we bias our score towards correctness of ingredients by assigning it a factor of 98% in the final weighted sum.

Due to the fact that the initial ingredients might have been combined in multiple ways before being used in a final step to create the dish, we have to backtrack in the simulated cooking process to determine which ingredients and how much of them is actually present in a dish. We call this operation 'dish unfolding'. This dish unfolding is a recursive operation in which we start from a dish and check what direct components it is comprised of and in which quantities. This process is repeated until we reach the base ingredients. This way we can determine how much of each base ingredient is present in a dish, while also knowing the sequence of intermediate food products they have been combined into. We start the comparison from the base ingredients instead of the final food products, because it is possible that a bowl contains all loose base ingredients and just one mixing operation is missing. This is still very close to the final dish which would be missed if we would just compare the final food products directly one at a time.

Once we have unfolded a predicted and a gold standard dish into their base ingredients, we look for the predicted base ingredient that maximally matches a gold standard base ingredient. This maximal matching is determined by a weighted sum of awarded points as well. Each property an ingredient has in common, e.g., being baked or having the right temperature, awards one point. The amount of each ingredient that should

be present is also seen as a property. This choice was made because the level of detail in our simulator currently does not suffice to reliably estimate how impactful having more or less of an ingredient would be. Therefore, we consider quantity mistakes to be general property mistakes as well. The points ratio for these ingredient properties are then computed and get a weight of 60%. In addition to these ingredient properties, we also check the sequence of intermediate food products they are used in and how many properties these intermediate food products have in common. The similarity of their intermediate food products gets a weight of 40%, which is relatively high but was chosen as certain combination operations do seem to have a big impact on the taste of their base ingredients. Since it is not guaranteed that a gold standard base ingredient is actually present in the predicted base ingredients, we compute this similarity score for each pair of ingredients and greedily choose the one with the highest score.

Each of these base ingredients will have an equally weighted influence on the 98% of the total dish score that is based on ingredient correctness. Gold standard ingredients that are missing or predicted ingredients that are in excess will each be considered to be an ingredient for which a similarity score of zero was achieved.

The aforementioned comparison process could be improved given an even more detailed, physics-based simulator. However, this dish approximation score does already seem to provide a useful additional metric for gaining insight into model results. A clarifying practical example is given in the following paragraphs and a pseudo-algorithm for computing the approximation dish score can be found in Appendix D.

Computation Example of Dish Approximation Score

In the following paragraphs we will manually compute the dish approximation score on a simplified dish in order to further clarify the ideas explained earlier. A graphical representation of the two dishes that will be compared is given in Figure 3.5.

The first 2% of the dish approximation score depends on properties of the dish that are not directly related to its actual contents. This includes location and container properties. As can be seen in Figure 3.5, most container properties are the same. Both containers are used, lined with baking paper, filled up from side to side and are located on the kitchen's countertop. However, the container itself is from a different type since the gold standard expects a baking tray and the predicted dish is served on a cookie sheet. Additionally, the number of portions that have been made differ as well, with the gold standard and predicted dish respectively having 25 and 20 portions. This leads to the following score computation, with four out of six possible points being awarded:

$$score_{container} = \frac{1+1+1+1+0+0}{1+1+1+1+1+1} = 0.67 \tag{3.2}$$

Next, the actual contents of the dish will be compared. This comparison will be preceded by a 'dish unfolding' process, which is demonstrated in Figure 3.6 for the gold standard dish. Each portion is first unfolded into base ingredients, after which we merge together base ingredients that overlap completely except for their available quantities. This simplifies the further comparison process and lowers the impact of proportioning

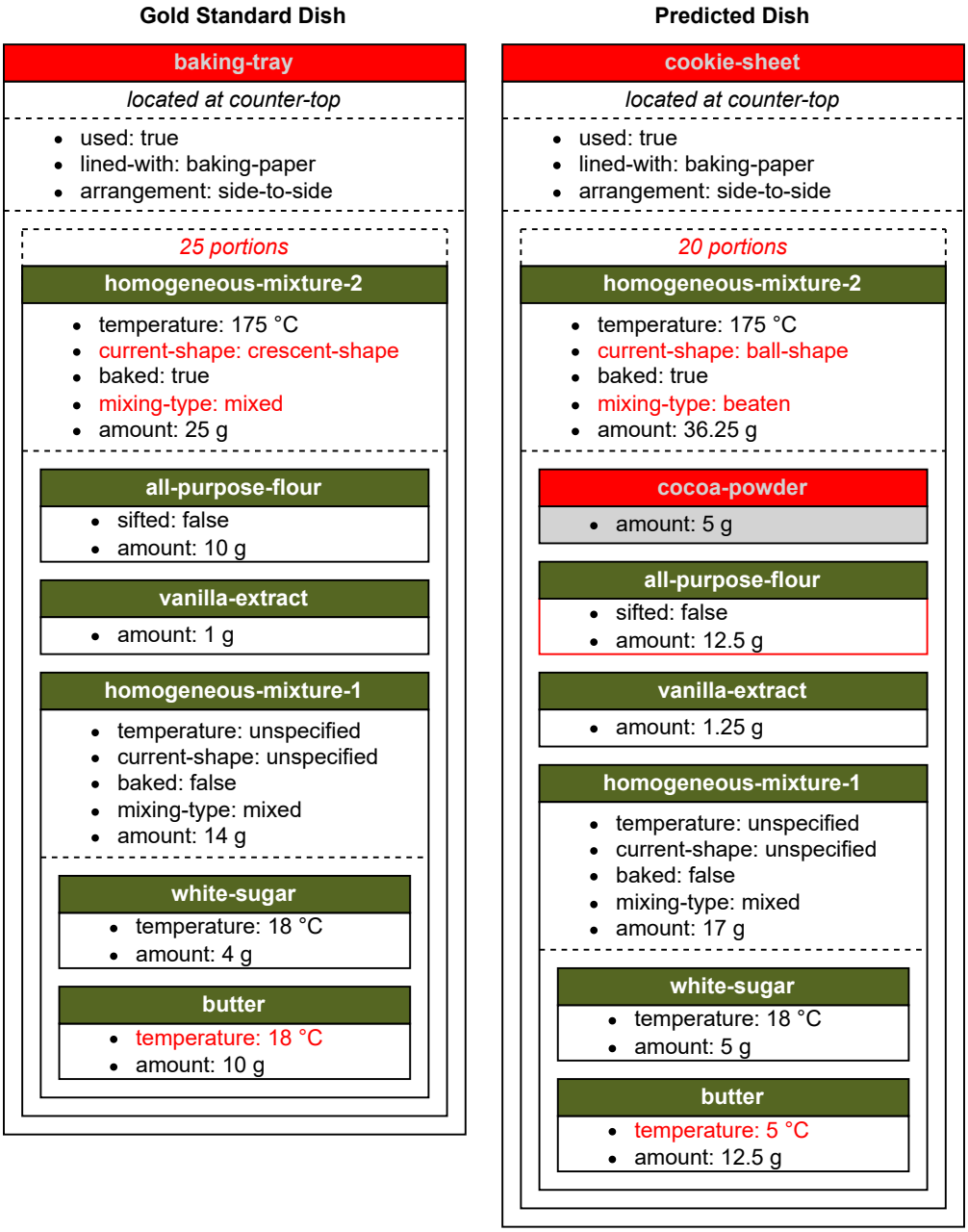

Figure 3.5: Graphical representation of a gold standard dish and a predicted dish. Differences between them that will influence the dish approximation score have been highlighted in red. Although the amounts often differ, this will have no direct influence as this difference is actually caused by a difference in number of portions.

errors, since the amounts of every single portion could be incorrect solely because the wrong number of portions have been made. However, the amounts of each base ingredient after merging still have to coincide for the gold standard dish and predicted dish in order to not be punished further as a mistake.

It is also important to note here that complete overlap is needed in order to be mergeable. If two base ingredients have some properties that differ or belong to different mixture hierarchies, they are considered to be two different ingredients even if they have the same ingredient type. Because such a case is less common, this has not been included in this example for simplicity sake. However, all concepts generally remain the same except in that case we will later rely on a greedy search on achieved ingredient comparison scores to find the two base ingredients that are most similar since we cannot rely solely on ingredient type.

In the example of Figure 3.5, each ingredient type only occurs once and the amounts of each base ingredient is the same for both dishes after merging the portions. As can be seen in Figure 3.6 the gold standard dish has four base ingredients, while the predicted dish will have one more due to the addition of cocoa powder. Since mixture hierarchies are recurring for these base ingredients, we will first compute a score for them. The *all-purpose-flour* and *vanilla-extract* are only part of *homogeneous-mixture-2*. Therefore, only the overlap between *homogeneous-mixture-2* of both dishes have to be taken into account. The temperature and state of being baked are the same, but the shape and mixing type differ. Moreover, both are the same type of mixture. This leads to the following score computation:

$$score_{mixture-2} = \frac{1+1+0+0+1}{1+1+1+1+1} = 0.60$$

The base ingredients *white-sugar* and *butter* are part of a sequence composed of *homogeneous-mixture-1* and *homogeneous-mixture-2*. There is complete overlap in properties for *homogeneous-mixture-1* and each mixture has equal weight in the computation of sequence correctness. This leads to the following score computation:

$$score_{mixture-1,2} = \frac{1 + score_{mixture-2}}{2} = 0.8$$

For the base ingredients *all-purpose-flour* and *vanilla-extract* there is complete overlap in properties between the two dishes and the mixture sequence score will be based on $score_{mixture-2}$. Giving a weight of 60% and 40% to the property overlap and mixture sequence score respectively, we get the following score computation:

$$score_{flour} = score_{vanilla} = 0.60 \times 1 + 0.40 \times score_{mixture-2} = 0.84$$

For the base ingredient *white-sugar* there is complete overlap in properties between the two dishes as well, but the mixture sequence score will be based on $score_{mixture-1,2}$. Giving a weight of 60% and 40% to the property overlap and mixture sequence score respectively, we get the following score computation:

$$score_{sugar} = 0.60 \times 1 + 0.40 \times score_{mixture-1-2} = 0.92$$

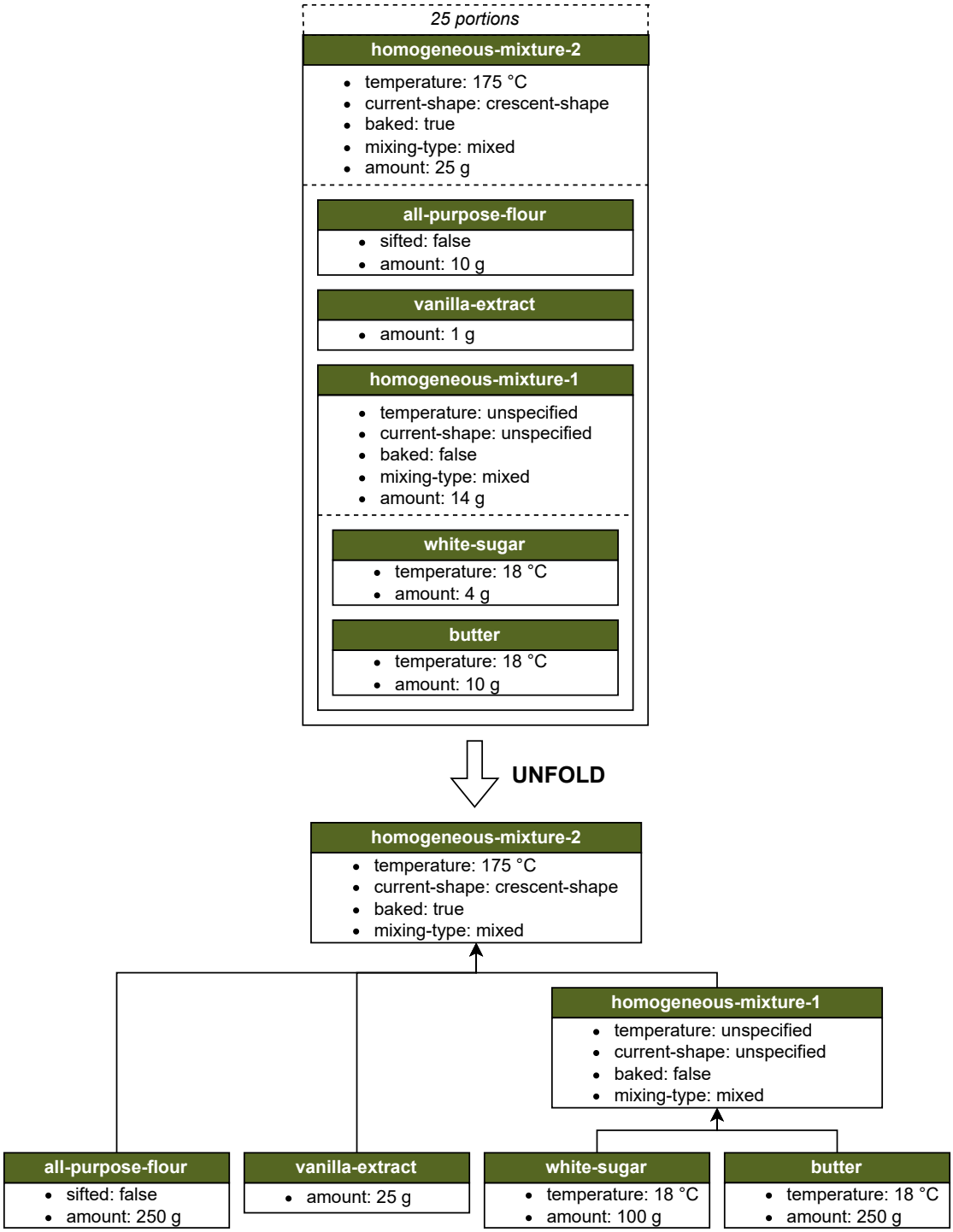

Figure 3.6: Graphical representation of the dish unfolding process in which 25 portions of a homogeneous mixture get unfolded into base ingredients.

For the base ingredient *butter* there is no complete overlap in properties between the two dishes, since their temperature differs. The amount, however, will be correct when taking the mistake in portions into account. The mixture sequence score will be based on $score_{mixture-1,2}$ and weights of 60% and 40% will be used as before, leading to the following score computation:

$$score_{butter} = 0.60 \times \frac{0+1}{1+1} + 0.40 \times score_{mixture-1-2} = 0.62$$

All base ingredients have an equal weight in the dish approximation score component based on contents, which leads to the following score computation:

$$score_{contents} = \frac{score_{flour} + score_{vanilla} + score_{sugar} + score_{butter} + 0}{5} = 0.64$$

The zero score addition follows from the presence of the base ingredient *cocoa-powder* in the predicted dish, since that ingredient is not present in the gold standard dish. Absent or excess ingredients will lead to the addition of zero scores which can have a significant influence on the final dish approximation score, as they can be expected to have a significant influence on real-world taste as well. The same approach is also taken for missing or excess steps in mixture sequences, with additional zero values being added there too. However, this is not the case for our example here.

The final dish approximation score can then be computed by performing a weighted sum of the scores obtained for the dishes' contents and container comparisons, with each respectively counting for 98% and 2% of the final score. This leads to the following final score computation:

$$dish\ approximation\ score = 0.02 \times score_{container} + 0.98 \times score_{contents} = 0.65$$

It should be noted that the explained computation was performed on a simplified version of the dishes. Both containers and base ingredients will generally have more properties that can be compared in our simulator, which causes some small mistakes to have less of an impact on the final score.

Recipe Execution Time

Recipe execution time, which is based on the time of availability as explained in section 3.4.2, checks how many simulation time steps it would take before the last argument would be available if we follow the predicted semantic network. We consider this to be the total execution time for the simulated cooking process. This metric is included because neither of the other two simulation-based metrics take cooking efficiency into account, while this could also be an indication of insight into what is needed for a recipe.

Other efficiency metrics could have been included as well, such as minimizing the number of dirty tools, but execution time seems to be the most all-encompassing one. Reusing tools when possible will also lead to a lower execution time for example compared to fetching and using different tools. Nevertheless, other efficiency metrics might be

included in future versions of the benchmark if such efficiency optimizations would be deemed useful.

Although recipe execution time can already be used to compare different solutions without knowing the gold standard execution time, an overview of the recipe execution times of all gold standard solutions have been included as a baseline in the benchmark's documentation for completeness sake. However, it should be noted that a lower execution time than the presented baseline does not automatically mean better performance as it could be caused by inadequate comprehension of the recipe and performing less or incorrect operations. Recipe execution time should thus always be interpreted in the context of other evaluation results.

3.5 Analysis of Benchmark Properties

Each benchmark will have to make trade-offs due to technical constraints or conflicting requirements. This means even a well-designed benchmark will have benefits and downsides associated with its use. Our benchmark is no exception to this rule, which is why in the next sections we will analyze its properties based on the important benchmark characteristics that were mentioned in section 2.2.3. Such an analysis and general transparency not only helps potential benchmark users to make the right choice for their use case, but also allows us to determine where potential improvements might be possible in the future.

3.5.1 Relevance

Maximizing relevance is one of the most important benchmark properties and is therefore one we focused on extensively. Our benchmark is aimed towards progressing a narrower context-specific task, namely recipe parsing for use in robotic cooking. Efforts to enhance transferability of benchmark performance to the real world are reflected in many made design choices. First, the representation language used in our benchmark is a domain-specific procedural language that can be directly used in robotic agents by providing an implementation for the primitives. Secondly, recipes have been chosen that contain many idiosyncrasies encountered in recipe texts which allows diagnosing models' strengths and weaknesses on common issues. Moreover, the quality of their annotations has been safeguarded by forgoing annotation via non-expert crowdsourcing in favor of annotation by computer science students with procedural knowledge. Additionally, these annotations have been validated through discussions guided by NLP experts and simulated execution testing. Thirdly, test set collection is separated from training set collection which could enhance generalization and does not overpromote the use of one particular type of model. Lastly, we provided metrics that are based on a cooking simulator to align benchmark performance measurements with real-world performance as much as possible.

Even though it is clear that increasing relevance has been a primary focus, there are also some benchmark downsides that should be addressed. First, ensuring annotations

were of an adequate quality and would be executable in simulation limited the number of recipes that are currently in our test set. Therefore, the current setup of our benchmark can be used to provide insight into problems and thus further progress the field but it should not be seen as a reliable indication of the status of the field overall. Secondly, the level of detail used in the simulator will affect the transferability of our performance measurements to the real world. Even though an abstraction level was chosen that already allows adequate measurements, using a more advanced simulator could lead to even more realistic measurements. Thirdly, our benchmark is built around Western recipes in English and is therefore biased towards them. This was needed as annotators and validators had to understand them, but results should thus not be considered representative for non-Western recipes or recipes written in other natural languages.

3.5.2 Reproducibility

Since semantic networks generally do not have an explicit execution order, IRL's search process could have some built-in stochasticity if multiple primitives would be eligible for execution at the same time. However, due to the fact that our primitives currently all have a unidirectional implementation and kitchen states are propagated throughout the network, execution order will in effect be deterministic. Moreover, our simulation-based metrics do not take execution order into account. Therefore, evaluation scores between different runs of the same model are consistent and reproducible.

Furthermore, our benchmark is mainly focused on research progression by providing an adequate and relevant evaluation environment. It does not have a leaderboard to prevent turning the benchmark into an effort to win a contest instead of maximally surveying the research domain in order to be as impactful as possible. Therefore, evaluation results can be expected to be published in research papers that describe the model in detail which also facilitates independent reproduction.

3.5.3 Fairness

Many design decisions have been made to augment fairness for all benchmark participants. No limitations have been put on the type of model that can be used, nor did we provide training data that would promote the use of any specific approaches. Furthermore, the proposed tools and methods are portable and thoroughly described in both this thesis and the benchmark's accompanying documentation.

Test data has been made publicly available to participants which potentially allows overfitting and perhaps abuse of prior knowledge. This choice, however, is made as a consequence of two other design decisions. First, we did not want to include a leaderboard to lessen the competitive urge that might lead to intentional or unintentional benchmark overfitting. Secondly, we wanted to make our simulator open source to allow independent future extensions. However, this means users would have access to the built-in solutions if they know where to look for them. Therefore, the fairer option was decided to be openly giving them to all participants. This access to the solutions is not necessarily a problem, since our benchmark focuses on being a testbed through which

different approaches can show their benefits and downsides to fully explore the research domain. Ideas behind developed approaches are considered more valuable than actually obtaining the highest possible evaluation scores.

3.5.4 Verifiability

Due to our benchmark setup, results will mostly be published in research papers detailing the general approach and model insights. Verifiability is thus satisfied from an academic perspective. From an evaluation perspective, benchmark results are obtained from data in an automated deterministic manner which means they can be considered completely accurate in that way as well.

3.5.5 Usability

The benchmark has portable, publicly available software tools that support the development and evaluation of models. These tools are easy to use and are well-documented in this thesis and the benchmark's accompanying documentation. The documentation addresses both result interpretation and technical use of the tools. Furthermore, all simulation and evaluation tools are open source allowing benchmark designers and users to extend them if needed.

The lightweightedness of the tools and the small dataset size should not lead to any high expenses for model evaluation, but the absence of training data might require bigger resource investments being needed for training data curation. This trade-off, however, was needed to improve other benchmark properties such as relevance and fairness by potentially enhancing generalization and avoiding overpromotion of specific model types.

Chapter 4

Tackling the Benchmark

This chapter describes the general approach to using our new benchmark for recipe understanding that has been presented in Chapter 3. Successful benchmark usage does not only imply being able to set it up and use it from a technical perspective, but also implies having insight into some of the linguistic and extralinguistic challenges present in the recipes provided with our benchmark. Therefore, both of these aspects will be addressed in this chapter.

We will first focus on giving more details regarding benchmark setup and usage from a technical standpoint in section 4.1, as such knowledge is important for ensuring adoption. Next, section 4.2 will briefly go over some practical considerations that should be taken into account when interpreting metric results. Furthermore, in this section we will also use examples to accentuate the advantage of combining multiple metrics. Finally, section 4.3 will delve deeper into some more complicated challenges that models will need to take into account in order to be successful. These challenges will be demonstrated using examples from the recipes in our benchmark. However, we will only bring to light the potential challenges without explicitly giving a solution for them to avoid promoting certain techniques or methodologies for tackling them.

4.1 Setup & Usage of Evaluation Tools

In this section we will describe how to set up an environment for running the benchmark simulator enabling our full evaluation system. As explained in section 3.4.3 simulation-based metrics are recommended as they can be very beneficial for measuring performance, but it is still possible to compute Smatch scores by only using our modified Python Smatch library. We will briefly explain how the latter can be done in section 4.1.1, after which we will delve deeper into the setup and usage of our simulator for model evaluation in sections 4.1.2 and 4.1.3.

4.1.1 Python Library Smatch

Our modified Python library for computing Smatch scores has been made available as a standalone tool[1]. It only requires a Python version 3.5 or higher and can be used without any additional setup. The same tool is integrated into our simulator, but the library can also be used independently with the provided gold standard solution files and with gold standard solutions that have been made by users themselves.

Although its primary use is for computing the Smatch score, which is a special kind of F-score as explained in section 3.4.3, the library also supports computing precision and recall scores as they are both components used in the F-score computation. The meaning networks of the prediction and the gold standard solution, for which the score should be computed, can be passed in string format. Alternatively, paths to files that each contain one of the two aforementioned meaning networks are supported as well. This alternative is generally recommended for larger networks, which will be the case for many of the recipes in our benchmark.

The following console commands will compute the Smatch score for a predicted meaning network of `(pred-1 ?x)` and an expected meaning network of `(pred-1 ?x) (pred-2 ?x)` with the networks respectively being passed as string or file arguments:

```
python smatch.py -m "(pred-1 ?x)" "(pred-1 ?x) (pred-2 ?x)"
python smatch.py -m "path_1/file_1.txt" "path_2/file_2.txt"
```

Adding the argument *–pr* to these commands will additionally include the precision and recall in the output. The order in which the predicted and expected meaning network are passed does not matter when computing the Smatch score as the F-score is symmetric, but argument order does matter for precision and recall scores.

4.1.2 Simulator Executable

The benchmark simulator, with its built-in evaluation tools, is made available in the form of a standalone executable that allows one-click runs without needing prior installation. This executable can either be called via command-line with the appropriate arguments or it can be run directly in which case a graphical interface is shown requesting the user to specify the arguments. The former is expected to be used the most for actual training and development of models, while the latter has been added to enhance user-friendliness. Having a graphical user interface allows users to easily test out the system manually or to quickly select a particular solution for further analysis.

Command-line Runs

Command-line runs with the executable have two mandatory arguments that specify an input and output file path, namely *-input* and *-output* respectively. Additionally, there are multiple optional arguments to further fine-tune the evaluation system. An overview

[1] https://github.com/RobinDHVUB/smatch

of all options is given in Table 4.1. An execution call that includes all these possible options for an evaluation run on Windows would look as follows:

```
cookingbot-evaluator.exe -input "input_file_path/predictions.solution"
                         -output "output_file_path/results.csv"
                         -show-output true
                         -metrics smatch-score
                                  goal-condition-success
                                  dish-approximation-score
                                  execution-time
                         -lib-dir "smatch_dir_path"
```

Argument	**Description**
-input	Path to a .solution file with predicted semantic networks
-output	Path to a .csv file to write evaluation results to
-show-output	If *true* or *t* the simulation process is visualized in the browser
-metrics	Evaluation metrics to use: *smatch-score*, *goal-condition-success*, *dish-approximation-score*, *execution-time* or *none*
-lib-dir	Path to the Smatch library directory (required for *smatch-score*)

Table 4.1: An overview of all command-line arguments that are available for a *cookingbot-evaluator* call.

The input file path specified by *-input* should be a path to a .solution file containing one or more semantic network predictions that should be evaluated. Each semantic network should be prefixed by its recipe ID, which has been provided via the ID field in the structured recipe text as explained in section 3.3.2. Redundant whitespace or single-line comments via *;* present in the solution file are ignored during the parsing process. Many example solution files are provided in the benchmark's accompanying documentation and a simple example is also shown in Figure 4.1.

After parsing the semantic networks, they are first checked for syntactic correctness. Next, they are executed in the simulator and the requested metrics are computed by comparing each prediction to the gold standard. Finally, the evaluation results are written to a CSV file specified by the output file path given via *-output*. An example extract of such a CSV file is shown in Figure 4.2.

Additionally, an optional argument *-show-output* can be provided and is expected to be a boolean specifying whether Babel's interactive web visualization should be used to visualize the predicted network's execution for further analysis. If this argument is *true*, a web browser will be automatically opened at http://localhost:8000 to elucidate the simulation process of which a visualization was shown earlier in Figure 3.4. For efficiency reasons, the predicted network's execution is not visualized by default as visualization comes with overhead costs.

Lastly, a list of evaluation metrics can be given via the optional argument *-metrics*.

```
#almond-crescent-cookies
(get-kitchen ?kitchen)
(fetch-and-proportion ?proportioned-butter ?ks-with-butter
     ?kitchen ?target-container-1 butter 230 g)

; IMPLICIT: let the butter warm up
(bring-to-temperature ?warm-butter ?ks-with-warm-butter
     ?ks-with-butter ?proportioned-butter 18 degrees-celsius)

(fetch-and-proportion ?proportioned-sugar ?ks-with-sugar
     ?ks-with-warm-butter ?target-container-2 white-sugar 120 g)
(transfer-contents ?output-container-a ?rest-a ?output-ks-a
     ?ks-with-sugar ?empty-container-a ?warm-butter
     ?quantity-a ?unit-a)
(transfer-contents ?output-container-b ?rest-b ?output-ks-b
     ?output-ks-a ?output-container-a ?proportioned-sugar
     ?quantity-b ?unit-b)
(beat ?beaten-mixture ?ks-with-beaten-mixture
     ?output-ks-b ?output-container-b ?mixing-tool)
(bake ?baked-crescents ?ks-with-baked-crescents
     ?ks-with-beaten-mixture ?beaten-mixture ?oven
     15 minute 175 degrees-celsius)

#easy-banana-bread
(get-kitchen ?kitchen)
(fetch-and-proportion ?proportioned-eggs ?ks-with-eggs
     ?kitchen ?target-container-1 egg 2 piece)
(fetch-and-proportion ?proportioned-bananas ?ks-with-bananas
     ?ks-with-eggs ?target-container-2 banana 3 piece)
(mash ?mashed-bananas ?ks-with-mashed-bananas
     ?ks-with-bananas ?proportioned-bananas ?fork)
(transfer-contents ?output-container-a ?rest-a ?output-ks-a
     ?ks-with-mashed-bananas ?empty-container-a ?proportioned-eggs
     ?quantity-a ?unit-a)
(transfer-contents ?output-container-b ?rest-b ?output-ks-b
     ?output-ks-a ?empty-container-b ?mashed-bananas
     ?quantity-b ?unit-b)
(beat ?batter ?ks-with-batter
     ?output-ks-b ?output-container-b ?beating-tool)
(bake ?baked-banana-bread ?ks-with-baked-banana-bread
     ?ks-with-batter ?batter ?oven
     60 minute 165 degrees-celsius)
```

Figure 4.1: An example solution file containing a predicted solution for the 'Almond Crescent Cookies' and the 'Easy Banana Bread' recipes. It should be noted that these are not the gold standard solutions, but rather random incorrect solutions demonstrating the general structure of a solution file.

```
recipe-id,smatch-score,dish-approximation-score,execution-time
easy-banana-bread,0.51,0.14,3820
almond-crescent-cookies,0.43,0.24,1830
```

Figure 4.2: A CSV file containing the evaluation results of a *cookingbot-evaluator* call with *smatch-score*, *dish-approximation-score* and *execution-time* as requested metrics. The results shown here are the evaluation results for the solution file from Figure 4.1.

If no metrics are specified, the system defaults to evaluation using the three simulation-based metrics. Possible arguments for *-metrics* are:

- *smatch-score*: compute the Smatch score
- *goal-condition-success*: compute the goal-condition success score
- *dish-approximation-score*: compute the dish approximation score
- *execution-time*: keep track of the recipe execution time
- *none*: do not compute any metrics, which can be useful if only the recipe's simulated execution visualization would be desired for example

In case *smatch-score* is specified as a metric that should be used, then the optional argument *-lib-dir* becomes required. This argument specifies where the Python Smatch library from section 4.1.1 is located. This library is also bundled together with the benchmark's simulator for convenience sake.

Graphical User Interface

If the executable is run directly, a graphical user interface is shown that requests the same information as would otherwise be given via command-line. It allows users to easily get acquainted with the different options by performing some quick experiments with the simulator and the evaluation metrics without having to write custom scripts for model development. As shown in figure 4.3 starting evaluation is disabled until at least the mandatory arguments, i.e., the input and output paths, are provided. Specifying the directory to the Smatch library is disabled as well until Smatch score evaluation is requested.

4.1.3 Babel Toolkit

In addition to the standalone executable, the benchmark has been made available as part of the Babel toolkit (Loetzsch et al., 2008; Nevens et al., 2019). This is an open source toolkit containing modules for implementing and running language- and communication-related experiments in simulated environments or real-world environments with physical robots. In contrast to the executable, Babel toolkit installation and setup is not a one-click process but it has the advantage of providing other language-related development tools as well as benchmark extensibility.

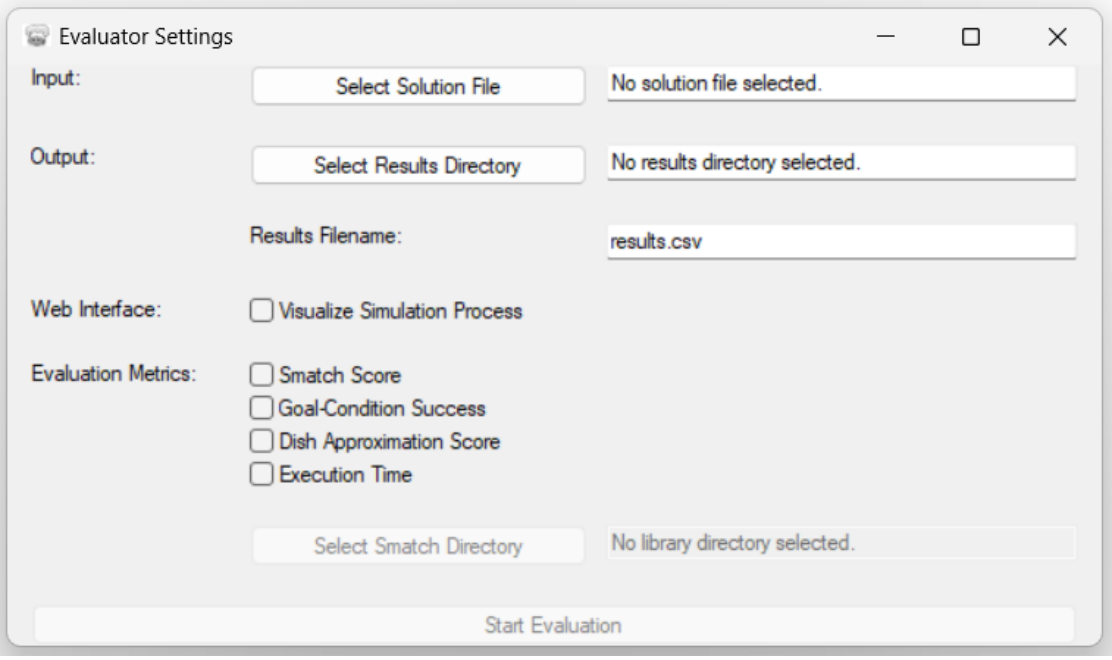

Figure 4.3: Graphical user interface that is shown when starting up the *cookingbot-evaluator* executable directly. The 'Start Evaluation' button is disabled until input and output paths are specified. Smatch library selection is disabled unless 'Smatch Score' is checked.

Installation

Installing the Babel toolkit has been thoroughly explained in its online documentation[2] for Linux, Mac OS and Windows.

Some steps mentioned in the installation guidelines are not strictly necessary, e.g., installing some semantic network plotting dependencies, but could be useful to further combine available functionalities in Babel with our benchmark for deeper analysis of the semantic networks. However, the minimal installation requirements for using our benchmark are fulfilled by performing the following three steps.

1. Clone or download Babel from https://gitlab.ai.vub.ac.be/ehai/babel;
2. Set up a Common Lisp programming environment (with CCL[3], SBCL[4] or LispWorks[5]);
3. Load the Babel initialization file *init-babel.lisp*, which is available in the main directory of the Babel directory obtained from the first step.

Usage

Once the installation steps mentioned in 4.1.3 are fulfilled, actually using the benchmark evaluation system consists of the following two main steps.

1. Load the *muhai-cookingbot* package from Babel using quicklisp:

   ```
   (ql:quickload :muhai-cookingbot)
   ```

2. Call the muhai-cookingbot's *internal-evaluate* function, which is also the function called internally in the benchmark executable of section 4.1.2.

The *internal-evaluate* function has the same mandatory and optional arguments as the executable. This means the arguments consist of a required input and output file path specification in combination with zero or more visualization and metric arguments. An *internal-evaluate* call that includes all possible options would look as follows:

```
(internal-evaluate "input_file_path/predictions.solution"
                   "output_file_path/results.csv"
                   :show-output t
                   :metrics '(smatch-score
                              goal-condition-success
                              dish-approximation-score
                              execution-time)
                   :lib-dir "lib_dir_path")
```

[2] https://emergent-languages.org/wiki
[3] https://ccl.clozure.com
[4] https://www.sbcl.org
[5] http://www.lispworks.com/

Extensions

The main advantage of using the Babel toolkit instead of the executable is the possibility of extending the evaluation tools. Just like other Babel modules the simulator's modules are open source components, which can be found in the applications package *muhai-cookingbot*. This not only allows the addition of new metrics, but more importantly allows the creation of new initial kitchen states.

Although the initial kitchen state that is currently provided can already cover many recipes, it is possible that certain recipes need specific tools or ingredients that are currently not available in the simulator's kitchen. In most cases adding such tools or ingredients can happen in a straightforward manner by simply adding a new item class to the simulator's ontology. Nevertheless, we did not automate such an extension process since complex relations and hierarchies can exist between items, as explained in section 3.4.1. Therefore, adding new items should happen in a well-though-out and detailed manner. Furthermore, letting system extensions occur via Babel has the additional benefit that even the MUHAI Cooking Language could be extended with new primitive cooking operations if needed.

4.2 Interpretation of Metrics

This section will briefly go over some practical considerations related to the evaluation metrics, while accentuating the advantage of combining multiple metrics. Ideally, at least all simulation-based metrics are used as they focus on performance from different perspectives. Therefore, intelligently combining results of all three of them will give a more correct view of performance. More elaborate examples regarding the interpretation of metrics can be found in Appendix F and in the benchmark's documentation, but the following paragraphs will already explain some of the most important details to be aware of when interpreting evaluation results.

Predicate Order The order in which the predicates are mentioned in the solution file actually have no influence on the Smatch score nor on any of the simulation-based metrics. Only the implicit graph that is created through argument sharing matters for the metrics. Smatch score is a graph-comparative metric by nature and the simulation-based results also depend on the implicit graph as this graph is what actually determines execution order of the recipe steps.

Even the actual execution order of predicates might not always influence all evaluation results. If the execution order is changed, the implicit graph is changed which means the Smatch score will always be lower. However, it is possible that all goal-conditions are still reached and that the same dish is prepared with the same execution time ensuring all other metrics are the same. Additionally, it is also possible some simulation-based scores remain the same while others do not. Recipe execution time might become longer if steps are executed in a way that forces the agent to spend more time waiting. This can for example happen if a food product is put in an oven at a later time than in the gold

standard, while other steps are dependent on this food product and can therefore not be executed. Another example could be that goal-condition success might differ because the order in which ingredients are added to a bowl has changed, while this does not affect the dish approximation score nor the recipe execution time. The reason goal-condition success might be different is that a goal-condition of having a bowl containing only the first ingredient might never be reached if you add another ingredient to that bowl before adding the first ingredient.

Timing of Mistakes The moment during execution at which mistakes occur can have an important impact on the obtained results. Smatch score will generally be less influenced by the timing of a mistake, since the position of the mistake in the implicit graph does not really affect this score. Comparably, the dish approximation score will generally also be less influenced by the timing of a mistake since only the effect of the mistake on the final dish matters in this case, which is often not timing-related. The timing of a mistake could lead to less or more delay for other steps in which case recipe execution time is affected, but this effect also depends heavily on the type of mistake that was made.

However, goal-condition success is very sensitive to the timing of mistakes in general. If even a small mistake happens early on in the cooking process, it could lead to a lot of subsequent goal-conditions failing. This is because goal-conditions are checked very strictly. They are either fulfilled or not. If an ingredient is altered slightly in one of the first steps before being added to a mixture, for example, this will lead to all future goal-conditions involving that mixture to fail since the mixture will never be completely correct when it comes to those goal-conditions. On the other hand, any alterations to the final dish for example will not influence goal-condition success as all goal-conditions have already been reached at that point. In that case the dish approximation score, however, will definitely be impacted.

Type of Mistakes The estimated impact of the type of a mistake is mostly only derivable through the dish approximation score. The dish approximation score will for example be much lower when using a completely incorrect ingredient compared to a smaller mistake such as forgetting to warm up butter before adding it to a mixture. Goal-condition success, however, will be similar in both cases since all goal-conditions involving that ingredient will fail in exactly the same way. Similarly, Smatch scores only detect a mistake is made but do not take into account what type of mistake occurred. As mentioned in the previous paragraph, both the timing and type of a mistake will have a combined effect on recipe execution time depending on the delays it causes.

Efficiency Neither goal-condition success nor dish approximation score will take efficiency into account. Constantly fetching new tools instead of reusing them for example will not prevent reaching all required goal-conditions, nor will it change the composition of a dish. However, the time needed for executing all steps will have changed, as will the

implicit graph. This means both the recipe execution time and the Smatch score will be affected.

In general, efficiency will only be measured through recipe execution time and potentially the Smatch score. However, it should also be noted that recipe execution time has little meaning without also taking other simulation-based scores into account. Not performing any cooking steps will lead to a very low execution time, but also very low scores for goal-condition success and dish approximation.

4.3 Challenges Inherent to Recipe Texts

As mentioned in sections 2.1.1 and 2.1.2, recipes have some linguistic, ontological and practical characteristics that can facilitate but also complicate processing and understanding them. Facilitation is mostly a consequence of the abundance of raw recipe data and the more imperative, objective and goal-oriented nature of their sentences with less subjective viewpoint dependencies. Complication is mostly a consequence of diverse linguistic phenomena related to referring expressions and the need for adequate contextual and world knowledge to allow reasonable understanding.

In this section we will focus a bit more on some of the challenges related to recipe understanding, using concrete examples from recipes of our benchmark. These practical examples aid in gaining better insight into the problems, which allows benchmark users to create better models. The given examples are not intended to be exhaustive, but should provide initial insights into the type of issues that might present themselves. Similar and slightly divergent examples of challenges, with varying degrees of difficulty, can be found throughout our benchmark's recipes. It should also be noted that we will abstain from giving concrete solutions as this might subconsciously or consciously narrow down model design for other researchers.

4.3.1 Coreferences

For reasons of redundancy avoidance and general brevity, recipes will often represent certain terms or complex combinations by shorter pronouns or generic noun phrases. Sentences in which these phenomena occur will always need contextual information in order to understand them. Therefore, recipes can generally only be understood when processed as a whole instead of a collection of separate sentences.

Under coreferential phenomena we categorize all occurrences of linguistic expressions in which there is a one-to-one mapping possible to the same real-world entity even though a different wording is used for it. The difficulty in the context of recipes is that many cooking actions lead to physical or chemical changes of food or tools by combination, separation or general alteration. This could lead to either the introduction of completely new entities or semantic distinction being required between two referring expressions, namely with one reference representing the original entity and the other representing a transformed form of that same entity. Historical and ontological knowledge about actions and their effects might thus be needed for correct understanding of a recipe. As a

further complicating factor, such coreferences also do not limit themselves to expressions occurring in a direct sequence of sentences since multiple other instructions might be intertwined between them.

As an example we consider an excerpt from the 'Afghan Biscuits' recipe, which is shown in Figure 4.4. In this single recipe multiple occurrences of the previously mentioned challenges can already be found.

The first coreferential challenge comes in the form of the need for the recognition of brands. The ingredient list contains a reference to 'Kellogg's' while a later instruction refers to the same entity with the expression 'corn flakes'. To resolve such a reference, ontological knowledge is needed about what type of food the brand Kellogg's might refer to.

A second challenge is present in the form of creamed 'butter and sugar' later being referred to as a new entity called 'butter mixture'. This implies recollection and knowledge are required of already created mixtures and their composition. This need for implicit knowledge is further made clear by the later instruction for the creation of 'icing', which is also referred to as 'mixture'. Moreover, the composition of this new 'icing' entity is only made clear after the 'icing' entity has already been introduced. This means both back and forward reference resolution might be needed.

Lastly, the combination of ingredients generates another new entity called 'dough' by adding more ingredients to the aforementioned 'butter mixture'. Such a 'dough' entity can be expected to have different compositions in different recipes or potentially even in the same recipe if even more ingredients get added to it later on. The rolling action then divides this singular entity 'dough' into multiple balls, which is then consequently referred to with the pronoun 'them'. After baking, each ball of dough is transformed and later referred to as a 'cookie'. Moreover, between these two references the instructions for icing preparation are inserted making the relatively easy strategy of searching for the entity mentioned in the preceding sentence inadequate. A more global approach seems needed that takes both sequential history and ontological knowledge into account.

4.3.2 Ellipses & Implications

Another phenomenon that frequently occurs in recipes in order to generate a concise list of instructions are ellipses, in which missing words or steps are expected to be derivable from linguistic context as well as common sense world knowledge.

Gaps created by missing words are called zero anaphora, which are generally easier to detect than missing steps as the presence of zero anaphora are often made clear by the absence of expected grammatical structures. However, it should be noted that easier detection of the gaps does not imply that detecting what they refer to is easy as well. For both types of ellipses linguistic and extralinguistic knowledge might be needed.

In addition to the implication of missing entities and steps, using contextual and common sense cooking knowledge might also be required to correctly interpret certain instructions even if the instructions themselves do not explicitly contain gaps in their wording.

```
<ingredient>
    300 grams unsweetened Kellogg's or something similar
</ingredient>

...

<instruction>
    Cream the butter and sugar until light and fluffy.
</instruction>
<instruction>
    Sift together the flour and cocoa powder and mix into butter mixture
        with a wooden spoon.
</instruction>
<instruction>
    Fold in corn flakes and do not worry if they crumble.
</instruction>
<instruction>
    Roll or press 1 1/2 teaspoons grams of the dough into balls and flatten
        them slightly.
</instruction>
<instruction>
    Place them about 5 cm apart on the baking sheet.
</instruction>
<instruction>
    Bake in the oven for 10 to 15 minutes.
</instruction>
<instruction>
    Remove from oven, and cool on a wire rack.
</instruction>
<instruction>
    Prepare the icing by combining the icing sugar, unsweetened cocoa powder, and water
        in a bowl.
</instruction>
<instruction>
    Mix well until mixture is free of lumps and of a creamy consistency
</instruction>
<instruction>
    Spoon a little icing on each cookie.
</instruction>
```

Figure 4.4: An excerpt taken from the 'Afghan Biscuits' recipe of our benchmark, demonstrating coreferential challenges that might be present. Related coreferences that refer to similar entities are highlighted in the same way.

To give an overview of these challenges we will consider excerpts from two recipes of our benchmark. The excerpts come from the 'Afghan Biscuits' and 'Easy Banana Bread' recipes, which are shown in respectively Figures 4.5 and 4.6.

```
<ingredient>
    200 grams unsalted butter, room temperature
</ingredient>

...

<instruction>
    Roll or press 1 1/2 teaspoons of the dough into balls and flatten them
        slightly.
</instruction>
<instruction>
    Place them about 5 cm apart on the baking sheet.
</instruction>
<instruction>
    Bake (the balls of dough) in the oven for 10 to 15 minutes.
</instruction>
<instruction>
    Remove (them) from oven, and cool (them) on a wire rack.
</instruction>
```

Figure 4.5: An excerpt taken from the 'Afghan Biscuits' recipe of our benchmark, demonstrating zero anaphora and implication challenges that might be present. Zero anaphora are presented in italic, while other parts requiring inferential operations are underlined or highlighted in bold.

The ingredient list from 'Afghan Biscuits' clearly demonstrates that implication derivation skills might be needed for correct understanding of recipes. There is a declarative statement mentioning '200 grams unsalted butter, room temperature', but an agent will need access to common sense knowledge and reasoning skills in order to derive what is meant from a procedural standpoint. This short declarative statement actually implies fetching unsalted butter, weighing it to obtain 200 grams and letting it warm up to room temperature either by waiting or by actively heating it. Additionally, some common sense knowledge might be needed to derive what is meant by 'room temperature'.

The list of instructions from 'Afghan Biscuits' also contains a less straightforward example of an ambiguous statement that would be interpreted correctly by humans, while being difficult for artificial agents. The recipe contains the imperative statement 'Roll or press 1 1/2 teaspoons of the dough into balls', for which a literal interpretation would be to use only one and a half teaspoons of the dough to make balls. Knowing it is the penultimate instruction in the recipe and a lot of dough would remain unused, humans would quite straightforwardly reason that this instruction actually implies balls of dough should be made repeatedly until all dough has been used. Such subtle disambiguations in meaning often require taking the instructional nature and context of recipes into account.

Furthermore, knowing that such separate balls of dough will often still be referred to as a singular entity is also knowledge that is useful for resolving zero anaphora in the next instructions. It is not explicitly mentioned what should be baked, removed and cooled but contextually it can be inferred that these would be 'the balls of dough'. Moreover, common sense knowledge is also needed to know that 'the baking sheet' should be placed in the oven in order to bake the 'the balls of dough' and not the balls directly. Such inferences seem rather straightforward for humans, but could actually require a lot of ontological knowledge and reasoning.

```
<ingredient>
    2 eggs
</ingredient>

...

<instruction>
    (Crack the eggs.)
    Cream together butter, eggs and sugar until smooth.
</instruction>
<instruction>
    Add bananas and vanilla (to the creamed mixture); beat (the mixture) well.
</instruction>
<instruction>
    Mix in flour (into the mixture).
</instruction>
<instruction>
    (Transfer batter into a greased pan and spread it evenly.)
    Bake (the batter) (in the oven) at 165°C for about 1 hour.
</instruction>
```

Figure 4.6: An excerpt taken from the 'Easy Banana Bread' recipe of our benchmark, demonstrating zero anaphora, missing steps and other implication challenges that might be present. Zero anaphora are presented in italic, missing steps are presented in bold italic and transformed entities that are focused on have been underlined.

More examples of zero anaphora and examples can be found in the 'Easy Banana Bread' excerpt. The arguments of most verbs are elided in this recipe, because a reader can imply them from the context and commonsense cooking knowledge. Important to note here as well is that the elided arguments in this recipe actually all refer to newly introduced entities that have not been explicitly mentioned before, which could increase the difficulty of detecting the right reference to fill the gap.

The 'Easy Banana Bread' excerpt also demonstrates the presence of missing steps, which are expected to be implied by the reader but would lead to recipe failure if skipped. First, the very common occurrence of adding eggs is present, in which the recipe does not specify that the gathered eggs should actually be cracked and the contents have to be added instead of the eggs as a whole. This also demonstrates the importance of taking state changes into account, since the same word 'eggs' is used to refer to transformed

versions of a similar entity. Secondly, the recipe does not explicitly state that the created batter should be spread over a pan before it can be put in the oven.

Additionally, this excerpt also exemplifies that a recipe does often not mention which kitchen commodity should be used to perform a cooking operation. An agent should have commonsense knowledge in order to know what can be used for creaming, beating, mixing and also baking.

Finally, the common idiosyncrasy of dropped determiners in instructions is present here as well. Even though no definite articles are used when specifying operations on ingredients, the entities mentioned in the instructions should be linked correctly to the ingredients from the ingredient list.

4.3.3 Other Challenges

Referential challenges are the most prominent difficulties that will be encountered during recipe parsing, but there are many other challenging phenomena that can occur as well. Since most of our recipes are user-generated, many slight variations and even inconsistencies can be found sporadically. These can vary from accidental misspellings and grammatical errors to using different writing styles which might confuse agents. The instruction "Roll or press 1 1/2 teaspoons of the dough into balls and flatten them" from the 'Afghan Biscuits' recipe and the instruction "Take generous tablespoons of the dough and roll it into a small ball" from the 'Almond Crescent Cookies' recipe, for example, both imply an action should be performed repeatedly but both also use a different grammatical number while talking about the intermediate and resulting entities. Another version of an 'Almond Crescent Cookies' recipe even makes the repetitive nature of the instruction explicit by stating "Working with 1 tablespoon of dough at a time, lightly roll and then shape it into a ball".

In addition to dealing with differences in writing styles, certain disambiguations related to choosing the right operations might be subtle but complex as well. This complexity might even increase in the future when the types of recipes that our benchmark supports grow in number. In the context of salads and baked goods for example, which are both present in our benchmark, mixing lettuce and tomatoes should be seen as quite a different operation when compared to mixing dough. Spreading butter on a pan or spreading potatoes on a pan generally also requires different tools and execution, which means they are not semantically the same.

Finally, as is common for user-generated content the recipes in our benchmark also contain some declarative information which could be ignored but might actually be useful to expand general cooking knowledge of a model. This declarative information could be more subjective, e.g., "I find that the longer this dish marinates the tastier it is!", to objective hints for execution such as "it will be slightly crumbly".

Chapter 5

Conclusions & Future Work

The goal of this thesis is to strive towards finding an answer to the research question: "How can we design a benchmark that can foster progress in the domain of natural language understanding with a focus on the deep understanding of an everyday human activity, in particular the execution of recipes written in natural language in a kitchen environment?". Therefore, this thesis starts with an elaborate background study to lay the foundations needed for developing a high-quality recipe understanding benchmark. This includes giving an overview of existing recipe parsing approaches and difficulties they face. In this context, a strong focus is put on representations that are suited for recipe execution with many of them being graph-based. Additionally, we formally define what a benchmark is and delve deeper into its components and properties. During this background study it has also been discovered that benchmarks have both many proven benefits and many potential pitfalls. Succumbing to some of these pitfalls might be the reason that existing benchmarks have failed to achieve widespread adoption in the research community and performance on real-world recipe execution tasks is still limited. Based on the aforementioned findings, we developed a new benchmark for recipe understanding where each design choice is clearly motivated and explained. This benchmark development is followed by a subsequent deeper analysis and general usage guidelines to maximize transparency and encourage not only community adoption but also further improvements.

Keeping all of this in mind, the remainder of this chapter is outlined as follows. In section 5.1 we give a discussion and some concluding insights related to the efforts and findings regarding the stated research question. Finally, section 5.2 reflects on questions that are still left unanswered and potential future improvements. Moreover, it mentions some related research that is already being performed which might benefit from the research presented in this thesis.

5.1 Conclusions

Our preliminary research for the development of a new benchmark for recipe understanding has made it clear that the use of benchmarks is a historically proven way to increase

progress in a field of research. Nevertheless, existing benchmarks related to natural language understanding in the context of recipe execution have not achieved widespread adoption. Although multiple issues exist within them, one potential cause that analysis has brought to light is the absence of a clear method for evaluating a model's performance on them. Therefore, evaluation methods have been a main point of focus in our new benchmark. Furthermore, our preliminary research led to the discovery of some potential pitfalls that are present in many recent artificial intelligence benchmarks. These include overfitting, oversaturation, overpromotion of a specific type of model and lack of transparency when it comes to benchmark design and applicability. Thus, avoiding these mistakes has been a second point of focus. At the same time, good properties that are present in existing benchmark have also been consistently taken into account. Such properties relate to benchmark relevance, reproducibility, verifiability, fairness and usability.

The design of our benchmark, which we named the MUHAI Recipe Execution Benchmark, clearly reflects the aforementioned points of focus. After deciding that our benchmark would consist of the concrete task of parsing a number of salad and baking recipes into a machine-readable and executable representation language, we developed the MUHAI Cooking Language. This is a formal graph-based language in which predicates correspond to executable primitive operations representing meaningful cooking-related actions, with predicate arguments corresponding to cooking-related concepts such as ingredients and tools. Its definition and design has been inspired by previous work in the field of recipe understanding, in which implicit and explicit graphs have proven to be useful as they can adequately capture relations, temporality and are both human- and machine-readable. Moreover, such representations have already been successfully used in robotic cooking experiments in the past, which proves their worth in the context of recipe execution.

After designing the MUHAI Cooking Language, most effort has been put into the design and development of evaluation tools because this was a shortcoming in existing recipe understanding benchmarks. Previous benchmarks either did not mention any metrics or used metrics that are not specific enough in the context of recipe understanding aimed at execution. Therefore, a new evaluation approach is considered that is simulation-based. A symbolic simulator supporting automatic execution and evaluation of parsed recipes is provided with our benchmark. This approach should ensure better transferability to real-world utility later on, since the goal of our benchmark requires recipe understanding that optimizes successful execution potential and not only understanding as many sentences as possible in isolation. Even though both goals have some overlap, the former requires more elaborate reasoning skills, memory and world knowledge.

To ensure better transferability and relevance of evaluation scores, a combination of multiple simulation-based metrics and a more traditional non-simulation-based metric has been deemed necessary as well. The four metrics that are provided are Smatch score, goal-condition success, dish approximation score and recipe execution time. The Smatch score has been included because it has been specifically developed for use as a

common metric for semantic structures and has known good adoption rates within our target community. However, the combination of the three simulation-based metrics is more specific to our goal as they can more closely gauge how close to successful recipe execution we are. Goal-condition success measures how many goal-conditions we have reached on our way to the final dish, which thus also estimates where execution goes wrong. The dish approximation score estimates the similarity between the created main dish and the gold standard dish, independent of how this dish was reached and whether certain goal-conditions were traversed. This allows for more deviations during execution as long as the final dish is correct. Lastly, from a practical standpoint, correct execution is not enough for good applicability which means an execution efficiency measure is needed. This led to the inclusion of recipe execution time as a metric.

In addition to providing usable and relevant evaluation tools, our benchmark needed to avoid the mistakes of overfitting, oversaturation and overpromotion of specific models. These points of focus are mainly visible through the composition of our dataset and the absence of a leaderboard, which deviate from choices made in many modern benchmarks. Modern benchmarks often provide large-scale sets of training data combined with leaderboards, which promotes the use of resource-intensive deep learning methods and even benchmark overfitting to achieve high rankings. In contrast, our benchmark is designed without a leaderboard or training data but instead only provides some examples, test data and ways to evaluate a model. This approach puts less emphasis on achieved scores and more emphasis on allowing different types of models to survey the domain fully and showcase the benefits of different approaches. To further accentuate the importance of analysis over mere performance, in this thesis we also shed some light on challenges present in many recipe texts such as various anaphora phenomena inherent to recipes.

At the same time, not providing training data does have the disadvantage of delivering less communal resources since benchmark users will have to put more resources into curating training data themselves. Trade-offs will always be needed in benchmark design, which is also the reason full transparency about design choices and benchmark applicability is important. We avoided the last mistake of lacking transparency by being as open as possible about every aspect from the design process. This includes mentioning both advantages and disadvantages as well as providing detailed information about the data collection process, the annotation process, the evaluation tools and the benchmark's composition and design in general.

In conclusion, this thesis offers an example of a benchmark that is specifically designed to help advance the domain of natural language understanding for recipe execution by autonomous agents. Whether the endeavor will be successful in the sense that such long-term goals will actually be achieved can not yet be determined, but this thesis nevertheless elucidates and illustrates development and analysis guidelines that are useful for future benchmark design.

5.2 Future Work

When it comes to future work related to this thesis, a first interesting choice would be to do a follow-up study focusing on adoption. A benchmark has been proposed that aims to garner community adoption and lead to domain progression, but the actual success of such an endeavor requires more time to have passed before it can actually be investigated. In addition, an extensive user survey could be performed that might bring to light certain deficits which could lead to further developments. However, in short-term research discovered deficits would probably mostly be related to only usability and user experience issues.

Furthermore, some known deficits exist that have been kept due to trade-offs that had to be made during development from a resource availability perspective. The transparency about these less optimal aspects of our benchmark and the open source availability hopefully leads to future improvements mitigating some of them. A first improvement would involve extensions to the dataset, simulator and potentially even the representation language in terms of covered recipe categories. The recipes that are currently supported consist of salads and baked goods which are sufficiently distinct to require a good combination of different approaches, but the cooking domain is still much larger than solely these types of dishes. Moreover, only English recipes are currently supported but prior research has shown that subtle and less subtle differences exist between recipes based on their language and culture. Broader support in that sense would be an interesting extension as well, but this would probably require a more elaborate annotation strategy to ensure both feasibility and quality of annotating.

Another interesting possible improvement would be the development of an even stronger, more realistic physics-based simulator. Since the evaluation is mostly based on simulations, a more advanced simulator should further enhance transferability of obtained performance to the real world. The ideas, concepts and the developed representation language of our benchmark could be reused with some modifications to be able to make maximal use of such a new simulator. At the moment of writing, the University of Bremen is actually actively performing research into the development of a physics-based virtual reality simulator for the designed representation language in collaboration with the Vrije Universiteit Brussel. Experiments with actual robotic execution could be interesting as well to test out transferability of performance, but success in such a setting would be dependent on many other components besides natural language understanding.

Pending a potential replacement of the simulator, another technical improvement could be the development of extension mechanisms for the standalone executable that do not require a Babel toolkit setup. The executable could be extended for example with paths to special-purpose files that contain custom kitchen states. This would potentially improve usability of the benchmark from an extensibility standpoint. However, since the target audience is expected to generally be technically proficient the current extension process via the Babel toolkit should not be problematic for our users either.

Finally, the main aim of any benchmark is actual use. So hopefully this benchmark inspires novel approaches for recipe understanding and potentially even leads to enough

domain progress that robotic recipe execution becomes viable. Furthermore, the detailed approach towards benchmark design and analysis outlined in this thesis might inspire other benchmark creators both inside and outside the field of natural language understanding.

Bibliography

Abacha, A. B., & Zweigenbaum, P. (2011). Medical Entity Recognition: A Comparison of Semantic and Statistical Methods. In K. B. Cohen, D. Demner-Fushman, S. Ananiadou, J. Pestian, J. Tsujii, & B. Webber (Eds.), *Proceedings of the BioNLP 2011 Workshop* (pp. 56–64). Association for Computational Linguistics (ACL).

Allen, J. (1995). *Natural Language Understanding* (2nd ed.). Benjamin-Cummings Publishing Company.

Badra, F., Bendaoud, R., Bentebibel, R., Champin, P.-A., Cojan, J., Cordier, A., Després, S., Jean-Daubias, S., Lieber, J., & Meilender, T. (2008). TAAABLE: Text Mining, Ontology Engineering, and Hierarchical Classification for Textual Case-Based Cooking. In M. Schaaf (Ed.), *Workshop Proceedings of the 9th European Conference on Case-Based Reasoning* (pp. 219–228). Springer-Verlag GmbH.

Barbosa-Silva, A., Ott, S., Blagec, K., Brauner, J., & Samwald, M. (2022). Mapping global dynamics of benchmark creation and saturation in artificial intelligence. *Nature Communications*, *13*, 1–11. https://doi.org/10.1038/s41467-022-34591-0

Barbu, A., Mayo, D., Alverio, J., Luo, W., Wang, C., Gutfreund, D., Tenenbaum, J., & Katz, B. (2019). ObjectNet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 9453–9463). Curran Associates, Inc.

Batra, D., Diwan, N., Upadhyay, U., Kalra, J. S., Sharma, T., Sharma, A. K., Khanna, D., Marwah, J. S., Kalathil, S., & Singh, N. (2020). RecipeDB: a resource for exploring recipes. *Database*. https://doi.org/10.1093/database/baaa077

Beetz, M., Klank, U., Kresse, I., Maldonado, A., Mösenlechner, L., Pangercic, D., Rühr, T., & Tenorth, M. (2011). Robotic Roommates Making Pancakes. In *Proceedings of the 11th IEEE-RAS International Conference on Humanoid Robots* (pp. 529–536). Institute of Electrical & Electronics Engineers (IEEE). https://doi.org/10.1109/Humanoids.2011.6100855

Blagec, K., Dorffner, G., Moradi, M., Ott, S., & Samwald, M. (2022). A global analysis of metrics used for measuring performance in natural language processing. In T. Shavrina, V. Mikhailov, V. Malykh, E. Artemova, O. Serikov, & V. Protasov (Eds.), *Proceedings of NLP Power! The 1st Workshop on Efficient Benchmarking in NLP* (pp. 52–63). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2022.nlppower-1.6

Blagec, K., Dorffner, G., Moradi, M., & Samwald, M. (2020). A critical analysis of metrics used for measuring progress in artificial intelligence. *arXiv preprint arXiv:2008.02577*. https://doi.org/10.48550/arXiv.2008.02577

Bollini, M., Tellex, S., Thompson, T., Roy, N., & Rus, D. (2013). Interpreting and Executing Recipes with a Cooking Robot. In J. P. Desai, G. Dudek, O. Khatib, & V. Kumar (Eds.), *Experimental Robotics: The 13th International Symposium on Experimental Robotics* (pp. 481–495). Springer-Verlag GmbH. https://doi.org/10.1007/978-3-319-00065-7_33

Borthwick, A. E. (1999). *A Maximum Entropy Approach to Named Entity Recognition* [Doctoral dissertation, New York University]. New York, NY, USA. https://cs.nyu.edu/media/publications/borthwick_andrew.pdf

Bosselut, A., Levy, O., Holtzman, A., Ennis, C., Fox, D., & Choi, Y. (2018). Simulating Action Dynamics with Neural Process Networks. *arXiv preprint arXiv:1711.05313*. https://doi.org/10.48550/arXiv.1711.05313

Bowman, S. R., & Dahl, G. E. (2021). What Will it Take to Fix Benchmarking in Natural Language Understanding? In K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tür, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, & Y. Zhou (Eds.), *Proceedings of the 2021 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 4843–4855). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2021.naacl-main.385

Buolamwini, J., & Gebru, T. (2018). Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In S. A. Friedler & C. Wilson (Eds.), *Proceedings of the 1st Conference on Fairness, Accountability and Transparency* (pp. 77–91). PMLR.

Cai, S., & Knight, K. (2013). Smatch: an Evaluation Metric for Semantic Feature Structures. In H. Schuetze, P. Fung, & M. Poesio (Eds.), *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics* (pp. 748–752). Association for Computational Linguistics (ACL).

Cambria, E., & White, B. (2014). Jumping NLP Curves: A Review of Natural Language Processing Research. *IEEE Computational Intelligence Magazine*, *9*(2), 48–57. https://doi.org/10.1109/MCI.2014.2307227

Chandu, K., Nyberg, E., & Black, A. W. (2019). Storyboarding of Recipes: Grounded Contextual Generation. In A. Korhonen, D. R. Traum, & L. Màrquez (Eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (pp. 6040–6046). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/P19-1606

Chandu, K. R., Dong, R.-P., & Black, A. (2020). Reading Between the Lines: Exploring Infilling in Visual Narratives. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing* (pp. 1220–1229). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2020.emnlp-main.93

Chang, M., Guillain, L. V., Jung, H., Hare, V. M., Kim, J., & Agrawala, M. (2018). RecipeScape: An Interactive Tool for Analyzing Cooking Instructions at Scale. In R. L. Mandryk, M. Hancock, M. Perry, & A. L. Cox (Eds.), *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (pp. 1–12). Association for Computing Machinery (ACM). https://doi.org/10.1145/3173574.3174025

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (pp. 1724–1734). Association for Computational Linguistics (ACL). https://doi.org/10.3115/v1/D14-1179

Chu, Y.-J., & Liu, T.-H. (1965). On the Shortest Arborescence of a Directed Graph. *Scientia Sinica*, *14*, 1396–1400.

Chuah, S. H.-W., Aw, E. C.-X., & Cheng, C.-F. (2022). A silver lining in the COVID-19 cloud: examining customers' value perceptions, willingness to use and pay more for robotic restaurants. *Journal of Hospitality Marketing & Management*, *31*(1), 49–76. https://doi.org/10.1080/19368623.2021.1926038

Chung, Y.-j. (2012). Finding Food Entity Relationships Using User-Generated Data in Recipe Service. In X. Chen, G. Lebanon, H. Wang, & M. J. Zaki (Eds.), *Proceedings of the 21st ACM International Conference on Information and Knowledge Management* (pp. 2611–2614). Association for Computing Machinery (ACM). https://doi.org/10.1145/2396761.2398704

Conneau, A., & Kiela, D. (2018). SentEval: An Evaluation Toolkit for Universal Sentence Representations. In N. Calzolari, K. Choukri, C. Cieri, T. Declerck, S. Goggi, K. Hasida, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, S. Piperidis, & T. Tokunaga (Eds.), *Proceedings of the 11th International Conference on Language Resources and Evaluation* (pp. 1699–1704). European Language Resources Association (ELRA).

Cordier, A., Lieber, J., Molli, P., Nauer, E., Skaf-Molli, H., & Toussaint, Y. (2009). WikiTaaable: A semantic wiki as a blackboard for a textual case-based reasoning system. In C. Lange, S. Schaffert, H. Skaf-Molli, & M. Völkel (Eds.), *Proceedings of the 4th Semantic Wiki Workshop at the 6th European Semantic Web Conference* (pp. 88–101). CEUR-WS.

D'Amour, A., Heller, K., Moldovan, D., Adlam, B., Alipanahi, B., Beutel, A., Chen, C., Deaton, J., Eisenstein, J., & Hoffman, M. D. (2022). Underspecification presents challenges for credibility in modern machine learning. *Journal of Machine Learning Research*, *23*(226), 1–61.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society*, *39*(1), 1–22. https://doi.org/https://doi.org/10.1111/j.2517-6161.1977.tb01600.x

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Do-

ran, & T. Solorio (Eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 4171–4186). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/N19-1423

Dixon, L., Li, J., Sorensen, J., Thain, N., & Vasserman, L. (2018). Measuring and Mitigating Unintended Bias in Text Classification. In J. Furman, G. E. Marchant, H. Price, & F. Rossi (Eds.), *Proceedings of the 32nd AAAI/ACM Conference on AI, Ethics, and Society* (pp. 67–73). Association for Computing Machinery (ACM). https://doi.org/10.1145/3278721.3278729

Donatelli, L., Schmidt, T., Biswas, D., Köhn, A., Zhai, F., & Koller, A. (2021). Aligning Actions Across Recipe Graphs. In M. Moens, X. Huang, L. Specia, & S. W. Yih (Eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (pp. 6930–6942). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2021.emnlp-main.554

Donoho, D. (2017). 50 Years of Data Science. *Journal of Computational and Graphical Statistics*, *26*(4), 745–766. https://doi.org/10.1080/10618600.2017.1384734

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., & Gelly, S. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv preprint arXiv:2010.11929*. https://doi.org/10.48550/arXiv.2010.11929

Dotan, R., & Milli, S. (2020). Value-laden Disciplinary Shifts in Machine Learning. In M. Hildebrandt, C. Castillo, L. E. Celis, S. Ruggieri, L. Taylor, & G. Zanfir-Fortuna (Eds.), *Proceedings of the 2020 ACM Conference on Fairness, Accountability, and Transparency* (pp. 294–303). Association for Computing Machinery (ACM). https://doi.org/10.1145/3351095.3373157

Edmonds, J. (1967). Optimum Branchings. *Journal of Research of the National Bureau of Standards*, *71*(4), 233–240.

Elsweiler, D., Trattner, C., & Harvey, M. (2017). Exploiting Food Choice Biases for Healthier Recipe Recommendation. In N. Kando, T. Sakai, H. Joho, H. Li, A. P. de Vries, & R. W. White (Eds.), *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 575–584). Association for Computing Machinery (ACM). https://doi.org/10.1145/3077136.3080826

Ethayarajh, K., & Jurafsky, D. (2020). Utility is in the Eye of the User: A Critique of NLP Leaderboards. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing* (pp. 4846–4853). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2020.emnlp-main.393

Fang, B., Baldwin, T., & Verspoor, K. (2022). What does it take to bake a cake? The RecipeRef corpus and anaphora resolution in procedural text. In S. Muresan, P. Nakov, & A. Villavicencio (Eds.), *Findings of the Association for Computational Linguistics: ACL 2022* (pp. 3481–3495). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2022.findings-acl.275

Flannery, D., Miayo, Y., Neubig, G., & Mori, S. (2011). Training Dependency Parsers from Partially Annotated Corpora. In H. Wang & D. Yarowsky (Eds.), *Proceedings of the 5th International Joint Conference on Natural Language Processing* (pp. 776–784). Association for Computational Linguistics (ACL).

Gaillard, E., Lieber, J., & Nauer, E. (2017). Adaptation of TAAABLE to the CCC'2017 Mixology and Salad Challenges, adaptation of the cocktail names. In A. A. Sánchez-Ruiz & A. Kofod-Petersen (Eds.), *Proceedings of ICCBR 2017 Workshops (CAW, CBRDL, PO-CBR), Doctoral Consortium, and Competitions co-located with the 25th International Conference on Case-Based Reasoning* (pp. 253–268). CEUR-WS.

Ge, M., Ricci, F., & Massimo, D. (2015). Health-Aware Food Recommender System. In H. Werthner, M. Zanker, J. Golbeck, & G. Semeraro (Eds.), *Proceedings of the 9th ACM Conference on Recommender Systems* (pp. 333–334). Association for Computing Machinery (ACM). https://doi.org/10.1145/2792838.2796554

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., & Wichmann, F. A. (2020). Shortcut Learning in Deep Neural Networks. *Nature Machine Intelligence*, *2*(11), 665–673. https://doi.org/10.1038/s42256-020-00257-z

Gimpel, K., Schneider, N., O'Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J., & Smith, N. A. (2010). Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments. In *Proceedings of the 49th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 42–47). Association for Computational Linguistics (ACL).

Grishman, R., & Sundheim, B. M. (1996). Message Understanding Conference-6: A Brief History. In *Proceedings of the 16th Conference on Computational Linguistics* (pp. 466–471). Association for Computational Linguistics (ACL). https://doi.org/10.3115/992628.992709

Gruber, T. R. (1995). Toward Principles for the Design of Ontologies Used for Knowledge Sharing? *International Journal of Human-Computer Studies*, *43*(5-6), 907–928. https://doi.org/10.1006/ijhc.1995.1081

Gundersen, O. E., & Kjensmo, S. (2018). State of the Art: Reproducibility in Artificial Intelligence. In S. A. McIlraith & K. Q. Weinberger (Eds.), *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (pp. 1644–1651). AAAI Press. https://doi.org/10.1609/aaai.v32i1.11503

Haenlein, M., & Kaplan, A. (2019). A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. *California management review*, *61*(4), 5–14. https://doi.org/10.1177/0008125619864925

Hamada, R., Ide, I., Sakai, S., & Tanaka, H. (2000). Structural Analysis of Cooking Preparation Steps in Japanese. In K. Wong, D. L. Lee, & J. Lee (Eds.), *Proceedings of the 5th International Workshop on Information Retrieval with Asian Languages* (pp. 157–164). Association for Computing Machinery (ACM). https://doi.org/10.1145/355214.355237

Harashima, J., Ariga, M., Murata, K., & Ioki, M. (2016). A Large-scale Recipe and Meal Data Collection as Infrastructure for Food Research. In N. Calzolari, K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the 10th International Conference on Language Resources and Evaluation* (pp. 2455–2459). European Language Resources Association (ELRA).

Harashima, J., & Hiramatsu, M. (2020). Cookpad Parsed Corpus: Linguistic Annotations of Japanese Recipes. In S. Dipper & A. Zeldes (Eds.), *Proceedings of the 14th Linguistic Annotation Workshop* (pp. 87–92). Association for Computational Linguistics (ACL).

Harashima, J., Hiramatsu, M., & Sanjo, S. (2020). Calorie Estimation in a Real-World Recipe Service. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(8), 13306–13313. https://doi.org/10.1609/aaai.v34i08.7041

Harashima, J., Someya, Y., & Kikuta, Y. (2017). Cookpad Image Dataset: An Image Collection as Infrastructure for Food Research. In N. Kando, T. Sakai, H. Joho, H. Li, A. P. de Vries, & R. W. White (Eds.), *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 1229–1232). Association for Computing Machinery (ACM). https://doi.org/10.1145/3077136.3080686

Harashima, J., & Yamada, Y. (2018). Two-Step Validation in Character-Based Ingredient Normalization. In I. Ide & Y. Yamakata (Eds.), *Proceedings of the Joint Workshop on Multimedia for Cooking and Eating Activities and Multimedia Assisted Dietary Management* (pp. 29–32). Association for Computing Machinery (ACM). https://doi.org/10.1145/3230519.3230589

Hashimoto, A., Sasada, T., Yamakata, Y., Mori, S., & Minoh, M. (2014). KUSK Dataset: Toward a Direct Understanding of Recipe Text and Human Cooking Activity. In A. J. Brush, A. Friday, J. A. Kientz, J. Scott, & J. Song (Eds.), *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (pp. 583–588). Association for Computing Machinery (ACM). https://doi.org/10.1145/2638728.2641338

Henaff, M., Weston, J., Szlam, A., Bordes, A., & LeCun, Y. (2017). Tracking the World State with Recurrent Entity Networks. *arXiv preprint arXiv:1612.03969*. https://doi.org/10.48550/arXiv.1612.03969

Hennessy, J. L., & Patterson, D. A. (2017). *Computer Architecture - A Quantitative Approach* (5th ed.). Morgan Kaufmann Publishers.

Higashiyama, S., Utiyama, M., Sumita, E., Ideuchi, M., Oida, Y., Sakamoto, Y., Okada, I., & Matsumoto, Y. (2020). Character-to-Word Attention for Word Segmentation. *Journal of Natural Language Processing*, *27*(3), 499–530. https://doi.org/10.5715/jnlp.27.499

Huppler, K. (2009). The Art of Building a Good Benchmark. In R. O. Nambiar & M. Poess (Eds.), *Proceedings of the 1st Technology Conference on Performance Evaluation and Benchmarking* (pp. 18–30). Springer-Verlag GmbH. https://doi.org/10.1007/978-3-642-10424-4_3

Hutchinson, B., Rostamzadeh, N., Greer, C., Heller, K., & Prabhakaran, V. (2022). Evaluation Gaps in Machine Learning Practice. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency* (pp. 1859–1876). Association for Computing Machinery (ACM). https://doi.org/10.1145/3531146.3533233

Jermsurawong, J., & Habash, N. (2015). Predicting the Structure of Cooking Recipes. In L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, & Y. Marton (Eds.), *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 781–786). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/D15-1090

Jiang, Y., Zaporojets, K., Deleu, J., Demeester, T., & Develder, C. (2020). Recipe instruction semantics corpus (RISeC): Resolving semantic structure and zero anaphora in recipes. In K. Wong, K. Knight, & H. Wu (Eds.), *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing* (pp. 821–826). Association for Computational Linguistics (ACL).

Jiang, Y., Zaporojets, K., Deleu, J., Demeester, T., & Develder, C. (2022). CookDial: a dataset for task-oriented dialogs grounded in procedural documents. *Applied Intelligence*, 1–19. https://doi.org/10.1007/s10489-022-03692-0

Jones, K. S. (1994). Natural Language Processing: A Historical Review. In A. Zampolli, N. Calzolari, & M. Palmer (Eds.), *Current Issues in Computational Linguistics: In Honour of Don Walker* (pp. 3–16). Springer Netherlands. https://doi.org/10.1007/978-0-585-35958-8_1

Kandel, S., Paepcke, A., Hellerstein, J. M., & Heer, J. (2012). Enterprise Data Analysis and Visualization: An Interview Study. *IEEE Transactions on Visualization and Computer Graphics*, *18*(12), 2917–2926. https://doi.org/10.1109/TVCG.2012.219

Kazama, M., Sugimoto, M., Hosokawa, C., Matsushima, K., Varshney, L. R., & Ishikawa, Y. (2018). A Neural Network System for Transformation of Regional Cuisine Style. *Frontiers in ICT*, *5*, 14–21. https://doi.org/10.3389/fict.2018.00014

Kiddon, C., Ponnuraj, G. T., Zettlemoyer, L., & Choi, Y. (2015). Mise en Place: Unsupervised Interpretation of Instructional Recipes. In L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, & Y. Marton (Eds.), *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 982–992). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/D15-1114

Kiddon, C., Zettlemoyer, L., & Choi, Y. (2016). Globally Coherent Text Generation with Neural Checklist Models. In J. Su, X. Carreras, & K. Duh (Eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (pp. 329–339). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/D16-1032

Kiela, D., Bartolo, M., Nie, Y., Kaushik, D., Geiger, A., Wu, Z., Vidgen, B., Prasad, G., Singh, A., Ringshia, P., Ma, Z., Thrush, T., Riedel, S., Waseem, Z., Stenetorp, P., Jia, R., Bansal, M., Potts, C., & Williams, A. (2021). Dynabench: Rethink-

ing Benchmarking in NLP. In K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tür, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, & Y. Zhou (Eds.), *Proceedings of the 2021 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 4110–4124). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2021.naacl-main.324

Kingsbury, P. R., & Palmer, M. (2002). From TreeBank to PropBank. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation* (pp. 1989–1993). European Language Resources Association (ELRA).

Kitagawa, Y., & Komachi, M. (2018). Long Short-Term Memory for Japanese Word Segmentation. In S. Politzer-Ahles, Y. Hsu, C. Huang, & Y. Yao (Eds.), *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation* (pp. 279–288). Association for Computational Linguistics (ACL).

Koch, B., Denton, E., Hanna, A., & Foster, J. G. (2021). Reduced, Reused and Recycled: The Life of a Dataset in Machine Learning Research. In J. Vanschoren & S. Yeung (Eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1* (pp. 1–13). NeurIPS.

Kudo, T., Yamamoto, K., & Matsumoto, Y. (2004). Applying Conditional Random Fields to Japanese Morphological Analysis. In D. Lin & D. Wu (Eds.), *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing* (pp. 230–237). Association for Computational Linguistics (ACL).

Kümmerer, M., Wallis, T. S., & Bethge, M. (2018). Saliency Benchmarking Made Easy: Separating Models, Maps and Metrics. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Proceedings of the 2018 European Conference on Computer Vision* (pp. 798–814). Springer International Publishing. https://doi.org/10.1007/978-3-030-01270-0_47

Lee, H., Shu, K., Achananuparp, P., Prasetyo, P. K., Liu, Y., Lim, E.-P., & Varshney, L. R. (2020). RecipeGPT: Generative Pre-Training Based Cooking Recipe Generation and Evaluation System. In A. E. F. Seghrouchni, G. Sukthankar, T. Liu, & M. van Steen (Eds.), *Companion Proceedings of the Web Conference 2020* (pp. 181–184). Association for Computing Machinery (ACM). https://doi.org/10.1145/3366424.3383536

Lenat, D. B. (1995). CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, *38*(11), 33–38. https://doi.org/10.1145/219717.219745

Li, J., Sun, A., Han, J., & Li, C. (2020). A Survey on Deep Learning for Named Entity Recognition. *IEEE Transactions on Knowledge and Data Engineering*, *34*(1), 50–70. https://doi.org/10.1109/TKDE.2020.2981314

Lin, A. S., Rao, S., Celikyilmaz, A., Nouri, E., Brockett, C., Dey, D., & Dolan, B. (2020). A Recipe for Creating Multimodal Aligned Datasets for Sequential Tasks. In D. Jurafsky, J. Chai, N. Schluter, & J. R. Tetreault (Eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 4871–4884). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2020.acl-main.440

Liu, A., Yuan, S., Zhang, C., Luo, C., Liao, Y., Bai, K., & Xu, Z. (2020). Multi-Level Multimodal Transformer Network for Multimodal Recipe Comprehension. In J. X. Huang, Y. Chang, X. Cheng, J. Kamps, V. Murdock, J. Wen, & Y. Liu (Eds.), *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 1781–1784). Association for Computing Machinery (ACM). https://doi.org/10.1145/3397271.3401247

Loetzsch, M., Wellens, P., De Beule, J., Bleys, J., & van Trijp, R. (2008). *The Babel2 Manual* (tech. rep. No. AI-Memo 01-08). AI-Lab VUB. Brussels, Belgium.

Maeta, H., Sasada, T., & Mori, S. (2015). A Framework for Procedural Text Understanding. In *Proceedings of the 14th International Conference on Parsing Technologies* (pp. 50–60). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/W15-2206

Malmaud, J., Huang, J., Rathod, V., Johnston, N., Rabinovich, A., & Murphy, K. (2015). What's Cookin'? Interpreting Cooking Videos using Text, Speech and Vision. In R. Mihalcea, J. Chai, & A. Sarkar (Eds.), *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 143–152). Association for Computational Linguistics (ACL). https://doi.org/10.3115/v1/N15-1015

Malmaud, J., Wagner, E., Chang, N., & Murphy, K. (2014). Cooking with Semantics. In Y. Artzi, T. Kwiatkowski, & J. Berant (Eds.), *Proceedings of the ACL 2014 Workshop on Semantic Parsing* (pp. 33–38). Association for Computational Linguistics (ACL). https://doi.org/10.3115/v1/W14-2407

Manning, C. D. (2011). Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? In A. F. Gelbukh (Ed.), *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing* (pp. 171–189). Springer-Verlag GmbH. https://doi.org/10.1007/978-3-642-19400-9_14

Marcinkiewicz, M. A. (1994). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313–330.

Marin, J., Biswas, A., Ofli, F., Hynes, N., Salvador, A., Aytar, Y., Weber, I., & Torralba, A. (2021). Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(1), 187–203. https://doi.org/10.1109/TPAMI.2019.2927476

Mason, M. (2018). Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, *1*(1), 1–28. https://doi.org/10.1146/annurev-control-060117-104848

McCann, B., Bradbury, J., Xiong, C., & Socher, R. (2017). Learned in Translation: Contextualized Word Vectors. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 6297–6308). Curran Associates, Inc.

Miller, G. A. (1995). WordNet: A Lexical Database for English. *Communications of the ACM*, *38*(11), 39–41. https://doi.org/10.1145/219717.219748

Mishra, B. D., Huang, L., Tandon, N., Yih, W., & Clark, P. (2018). Tracking State Changes in Procedural Text: a Challenge Dataset and Models for Process Paragraph Comprehension. In M. A. Walker, H. Ji, & A. Stent (Eds.), *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 1595–1604). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/n18-1144

Mohammadi, E., Naji, N., Marceau, L., Queudot, M., Charton, E., Kosseim, L., & Meurs, M.-J. (2020). Cooking Up a Neural-based Model for Recipe Classification. In N. Calzolari, F. Béchet, P. Blache, K. Choukri, C. Cieri, T. Declerck, S. Goggi, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the 12th International Conference on Language Resources and Evaluation* (pp. 5000–5009). European Language Resources Association (ELRA).

Mori, S., Maeta, H., Sasada, T., Yoshino, K., Hashimoto, A., Funatomi, T., & Yamakata, Y. (2014). FlowGraph2Text: Automatic Sentence Skeleton Compilation for Procedural Text Generation. In M. Mitchell, K. F. McCoy, D. D. McDonald, & A. Cahill (Eds.), *Proceedings of the 8th International Natural Language Generation Conference* (pp. 118–122). Association for Computational Linguistics (ACL). https://doi.org/10.3115/v1/W14-4418

Mori, S., Maeta, H., Yamakata, Y., & Sasada, T. (2014). Flow Graph Corpus from Recipe Texts. In N. Calzolari, K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the 9th International Conference on Language Resources and Evaluation* (pp. 2370–2377). European Language Resources Association (ELRA).

Mori, S., Sasada, T., Yamakata, Y., & Yoshino, K. (2012). A Machine Learning Approach to Recipe Text Processing. In *Proceedings of the 1st Workshop on Cooking with Computers* (pp. 1–6). Centre national de la recherche scientifique (CNRS).

Nadeau, D., & Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, *30*(1), 3–26. https://doi.org/https://doi.org/10.1075/li.30.1.03nad

Nakagawa, T. (2004). Chinese and Japanese Word Segmentation Using Word-Level and Character-Level Information. In *Proceedings of the 20th International Conference on Computational Linguistics* (pp. 466–472). Association for Computational Linguistics (ACL). https://doi.org/10.3115/1220355.1220422

Nanba, H., Doi, Y., Tsujita, M., Takezawa, T., & Sumiya, K. (2014). Construction of a Cooking Ontology from Cooking Recipes and Patents. In A. J. Brush, A. Friday, J. A. Kientz, J. Scott, & J. Song (Eds.), *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (pp. 507–516). Association for Computing Machinery (ACM). https://doi.org/10.1145/2638728.2641328

Neubig, G., Nakata, Y., & Mori, S. (2011a). Pointwise Prediction for Robust, Adaptable Japanese Morphological Analysis. In D. Lin, Y. Matsumoto, & R. Mihalcea

(Eds.), *Proceedings of the 49th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 529–533). Association for Computational Linguistics (ACL).

Neubig, G., Nakata, Y., & Mori, S. (2011b). Pointwise Prediction for Robust, Adaptable Japanese Morphological Analysis. In D. Lin, Y. Matsumoto, & R. Mihalcea (Eds.), *Proceedings of the 49th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 529–533). Association for Computational Linguistics (ACL).

Nevens, J., Van Eecke, P., & Beuls, K. (2019). A Practical Guide to Studying Emergent Communication through Grounded Language Games. In *Language Learning for Artificial Agents Symposium of the 2019 AISB Convention* (pp. 1–8). Society for the Study of Artificial Intelligence & the Simulation of Behaviour (AISB).

Nishimura, T., Ishiguro, K., Higuchi, K., & Kotera, M. (2022). Multimodal Dish Pairing: Predicting Side Dishes to Serve with a Main Dish. In Y. Yamakata, A. Hashimoto, & J. Chen (Eds.), *Proceedings of the 1st International Workshop on Multimedia for Cooking, Eating, and related APPlications* (pp. 1–9). Association for Computing Machinery (ACM). https://doi.org/10.1145/3552485.3554934

Nishimura, T., Tomori, S., Hashimoto, H., Hashimoto, A., Yamakata, Y., Harashima, J., Ushiku, Y., & Mori, S. (2020). Visual Grounding Annotation of Recipe Flow Graph. In N. Calzolari, F. Béchet, P. Blache, K. Choukri, C. Cieri, T. Declerck, S. Goggi, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the 12th International Conference on Language Resources and Evaluation* (pp. 4275–4284). European Language Resources Association (ELRA).

Oonita, T., & Kitayama, D. (2020). Extraction Method for a Recipe's Uniqueness Based on Recipe Frequency and LexRank of Procedures. In M. Indrawan-Santiago, E. Pardede, I. L. Salvadori, M. Steinbauer, I. Khalil, & G. Kotsis (Eds.), *Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications & Services* (pp. 241–245). Association for Computing Machinery (ACM). https://doi.org/10.1145/3428757.3429128

Pan, L.-M., Chen, J., Wu, J., Liu, S., Ngo, C.-W., Kan, M.-Y., Jiang, Y., & Chua, T.-S. (2020a). Multi-Modal Cooking Workflow Construction for Food Recipes. In C. W. Chen, R. Cucchiara, X. Hua, G. Qi, E. Ricci, Z. Zhang, & R. Zimmermann (Eds.), *Proceedings of the 28th ACM International Conference on Multimedia* (pp. 1132–1141). Association for Computing Machinery (ACM). https://doi.org/10.1145/3394171.3413765

Pan, L.-M., Chen, J., Wu, J., Liu, S., Ngo, C.-W., Kan, M.-Y., Jiang, Y., & Chua, T.-S. (2020b). Multi-Modal Cooking Workflow Construction for Food Recipes. In C. W. Chen, R. Cucchiara, X. Hua, G. Qi, E. Ricci, Z. Zhang, & R. Zimmermann (Eds.), *Proceedings of the 28th ACM International Conference on Multimedia* (pp. 1132–1141). Association for Computing Machinery (ACM). https://doi.org/10.1145/3394171.3413765

Papadopoulos, D. P., Mora, E., Chepurko, N., Huang, K. W., Ofli, F., & Torralba, A. (2022). Learning Program Representations for Food Images and Cooking Recipes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 16559–16569). Institute of Electrical & Electronics Engineers (IEEE). https://doi.org/10.1109/CVPR52688.2022.01606

Paullada, A., Raji, I. D., Bender, E. M., Denton, E., & Hanna, A. (2021). Data and its (dis)contents: A survey of dataset development and use in machine learning research. *Patterns*, *2*(11), 100336. https://doi.org/10.1016/j.patter.2021.100336

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (pp. 1532–1543). Association for Computational Linguistics (ACL). https://doi.org/10.3115/v1/D14-1162

Petrov, S., Das, D., & McDonald, R. (2012). A Universal Part-of-Speech Tagset. In N. Calzolari, K. Choukri, T. Declerck, M. U. Dogan, B. Maegaard, J. Mariani, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the 8th International Conference on Language Resources and Evaluation* (pp. 2089–2096). European Language Resources Association (ELRA).

Pini, A., Hayes, J., Upton, C., & Corcoran, M. (2019). AI Inspired Recipes: Designing Computationally Creative Food Combos. In R. L. Mandryk, S. A. Brewster, M. Hancock, G. Fitzpatrick, A. L. Cox, V. Kostakos, & M. Perry (Eds.), *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (pp. 1–6). Association for Computing Machinery (ACM). https://doi.org/10.1145/3290607.3312948

Prabhu, V. U., & Birhane, A. (2021). Large image datasets: A pyrrhic win for computer vision? In *Proceedings of the 2021 IEEE Winter Conference on Applications of Computer Vision* (pp. 1536–1546). Institute of Electrical & Electronics Engineers (IEEE). https://doi.org/10.1109/WACV48630.2021.00158

Raji, I. D., Denton, E., Bender, E. M., Hanna, A., & Paullada, A. (2021). AI and the Everything in the Whole Wide World Benchmark. In J. Vanschoren & S. Yeung (Eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1* (pp. 1–17). NeurIPS.

Rajpurkar, P., Jia, R., & Liang, P. (2018). Know What You Don't Know: Unanswerable Questions for SQuAD. In I. Gurevych & Y. Miyao (Eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics* (pp. 784–789). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/P18-2124

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). SQuAD: 100,000+ Questions for Machine Comprehension of Text. In J. Su, X. Carreras, & K. Duh (Eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (pp. 2383–2392). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/D16-1264

Ramshaw, L., & Marcus, M. (1995). Text Chunking using Transformation-Based Learning. In D. Yarowsky & K. Church (Eds.), *Proceedings of the 3rd Workshop on Very Large Corpora* (pp. 82–94). MIT Press.

Ratinov, L., & Roth, D. (2009). Design Challenges and Misconceptions in Named Entity Recognition. In S. Stevenson & X. Carreras (Eds.), *Proceedings of the 13th Conference on Computational Natural Language Learning* (pp. 147–155). Association for Computational Linguistics (ACL).

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (pp. 3982–3992). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/D19-1410

Ribeiro, J., Lima, R., Eckhardt, T., & Paiva, S. (2021). Robotic Process Automation and Artificial Intelligence in Industry 4.0 - A Literature review. *Procedia Computer Science*, *181*, 51–58. https://doi.org/10.1016/j.procs.2021.01.104

Ribeiro, R., Batista, F., Pardal, J. P., Mamede, N. J., & Pinto, H. S. (2006). Cooking an Ontology. In J. Euzenat & J. Domingue (Eds.), *Proceedings of the 12th International Conference on Artificial Intelligence: Methodology, Systems, and Applications* (pp. 213–221). Springer-Verlag GmbH. https://doi.org/10.1007/11861461_23

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., & Bernstein, M. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Sajadmanesh, S., Jafarzadeh, S., Ossia, S. A., Rabiee, H. R., Haddadi, H., Mejova, Y., Musolesi, M., Cristofaro, E. D., & Stringhini, G. (2017). Kissing Cuisines: Exploring Worldwide Culinary Habits on the Web. In R. Barrett, R. Cummings, E. Agichtein, & E. Gabrilovich (Eds.), *Proceedings of the 26th International Conference on World Wide Web Companion* (pp. 1013–1021). Association for Computing Machinery (ACM). https://doi.org/10.1145/3041021.3055137

Salvador, A., Drozdzal, M., Giró-i-Nieto, X., & Romero, A. (2019). Inverse Cooking: Recipe Generation From Food Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 10453–10462). Institute of Electrical & Electronics Engineers (IEEE). https://doi.org/10.1109/CVPR.2019.01070

Salvador, A., Hynes, N., Aytar, Y., Marin, J., Ofli, F., Weber, I., & Torralba, A. (2017). Learning Cross-Modal Embeddings for Cooking Recipes and Food Images. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3068–3076). IEEE Computer Society. https://doi.org/10.1109/CVPR.2017.327

Sanfeliu, A., & Fu, K.-S. (1983). A Distance Measure Between Attributed Relational Graphs for Pattern Recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*(3), 353–362. https://doi.org/10.1109/TSMC.1983.6313167

Sang, E. F., & De Meulder, F. (2003). Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In W. Daelemans & M. Osborne (Eds.), *Proceedings of the 7th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 142–147). Association for Computational Linguistics (ACL).

Sasada, T., Mori, S., Kawahara, T., & Yamakata, Y. (2015). Named Entity Recognizer Trainable from Partially Annotated Data. In K. Hasida & A. Purwarianti (Eds.), *Proceedings of the 14th International Conference of the Pacific Association for Computational Linguistics* (pp. 148–160). Springer Singapore. https://doi.org/10.1007/978-981-10-0515-2_11

Sato, T., Harashima, J., & Komachi, M. (2016). Japanese-English Machine Translation of Recipe Texts. In T. Nakazawa, H. Mino, C. Ding, I. Goto, G. Neubig, S. Kurohashi, I. H. Riza, & P. Bhattacharyya (Eds.), *Proceedings of the 3rd Workshop on Asian Translation* (pp. 58–67). Association for Computational Linguistics (ACL).

Schuster, M., & Paliwal, K. (1997). Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. https://doi.org/10.1109/78.650093

Seki, Y., & Ono, K. (2014). Discriminating Practical Recipes Based on Content Characteristics in Popular Social Recipes. In A. J. Brush, A. Friday, J. A. Kientz, J. Scott, & J. Song (Eds.), *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (pp. 487–496). Association for Computing Machinery (ACM). https://doi.org/10.1145/2638728.2641326

Shankar, S., Halpern, Y., Breck, E., Atwood, J., Wilson, J., & Sculley, D. (2017). No Classification without Representation: Assessing Geodiversity Issues in Open Data Sets for the Developing World. *arXiv preprint arXiv:1711.08536*. https://doi.org/10.48550/arXiv.1711.08536

Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., & Fox, D. (2020). ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 10737–10746). Institute of Electrical & Electronics Engineers (IEEE). https://doi.org/10.1109/CVPR42600.2020.01075

Simas, T., Ficek, M., Diaz-Guilera, A., Obrador, P., & Rodriguez, P. R. (2017). FoodBridging: A New Network Construction to Unveil the Principles of Cooking. *Frontiers in ICT*, *4*, 14. https://doi.org/10.3389/fict.2017.00014

Spranger, M., Pauw, S., Loetzsch, M., & Steels, L. (2012). Open-ended Procedural Semantics. In L. Steels & M. Hild (Eds.), *Language Grounding in Robots* (pp. 153–172). Springer US. https://doi.org/10.1007/978-1-4614-3064-3_8

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge Engineering: Principles and Methods. *Data & Knowledge Engineering*, *25*(1–2), 161–197. https://doi.org/10.1016/S0169-023X(97)00056-6

Suchanek, F. M., Kasneci, G., & Weikum, G. (2008). YAGO: A Large Ontology from Wikipedia and WordNet. *Journal of Web Semantics*, *6*(3), 203–217. https://doi.org/10.1016/j.websem.2008.06.001

Tasse, D., & Smith, N. A. (2008). *SOUR CREAM: Toward Semantic Processing of Recipes* (tech. rep. No. CMU-LTI-08-005). Carnegie Mellon University. Pittsburgh, PA, USA.

Tenorth, M., & Beetz, M. (2009). KNOWROB: Knowledge Processing for Autonomous Personal Robots. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 4261–4266). Institute of Electrical & Electronics Engineers (IEEE). https://doi.org/10.1109/IROS.2009.5354602

Tomori, S., Ninomiya, T., & Mori, S. (2016). Domain Specific Named Entity Recognition Referring to the Real World by Deep Neural Networks. In K. Erk & N. A. Smith (Eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (pp. 236–242). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/P16-2039

Twomey, N., Fain, M., Ponikar, A., & Sarraf, N. (2020). Towards Multi-Language Recipe Personalisation and Recommendation. In R. L. T. Santos, L. B. Marinho, E. M. Daly, L. Chen, K. Falk, N. Koenigstein, & E. S. de Moura (Eds.), *Proceedings of the 14th ACM Conference on Recommender Systems* (pp. 708–713). Association for Computing Machinery (ACM). https://doi.org/10.1145/3383313.3418478

Van den Broeck, W. (2008). Constraint-based compositional semantics. In A. D. M. Smith (Ed.), *The Evolution of Language: Proceedings of the 7th International Conference (EVOLANG7)* (pp. 338–345). World Scientific Press. https://doi.org/10.1142/9789812776129_0043

von Kistowski, J., Arnold, J. A., Huppler, K., Lange, K.-D., Henning, J. L., & Cao, P. (2015). How to Build a Benchmark. In L. K. John, C. U. Smith, K. Sachs, & C. M. Lladó (Eds.), *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering* (pp. 333–336). Association for Computing Machinery (ACM). https://doi.org/10.1145/2668930.2688819

Voutilainen, A. (2005). Part-of-speech tagging. In R. Mitkov (Ed.), *The Oxford Handbook of Computational Linguistics* (pp. 219–232). Oxford University Press. https://doi.org/10.1093/oxfordhb/9780199276349.013.0011

Wagstaff, K. (2012). Machine Learning that Matters. In *Proceedings of the 29th International Conference on Machine Learning* (pp. 529–536). Omnipress.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. (2019). SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 3266–3280). Curran Associates, Inc.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In T. Linzen, G. Chrupala, & A. Alishahi (Eds.), *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP* (pp. 353–355). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/W18-5446

Williams, A., Nangia, N., & Bowman, S. R. (2018). A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In M. A. Walker, H. Ji, & A. Stent (Eds.), *Proceedings of the 2018 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 1112–1122). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/N18-1101

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., & Funtowicz, M. (2020). Transformers: State-of-the-Art Natural Language Processing. In Q. Liu & D. Schlangen (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 38–45). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2020.emnlp-demos.6

Yagcioglu, S., Erdem, A., Erdem, E., & Ikizler-Cinbis, N. (2018). RecipeQA: A Challenge Dataset for Multimodal Comprehension of Cooking Recipes. In E. Riloff, D. Chiang, J. Hockenmaier, & J. Tsujii (Eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (pp. 1358–1368). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/D18-1166

Yamaguchi, Y., Inuzuka, S., Hiramatsu, M., & Harashima, J. (2020). Non-ingredient Detection in User-generated Recipes using the Sequence Tagging Approach. In W. Xu, A. Ritter, T. Baldwin, & A. Rahimi (Eds.), *Proceedings of the 6th Workshop on Noisy User-generated Text* (pp. 76–80). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2020.wnut-1.11

Yamakata, Y., Carroll, J., & Mori, S. (2017). A Comparison of Cooking Recipe Named Entities between Japanese and English. In I. Ide & Y. Yamakata (Eds.), *Proceedings of the 9th Workshop on Multimedia for Cooking and Eating Activities in conjunction with The 2017 International Joint Conference on Artificial Intelligence* (pp. 7–12). Association for Computing Machinery (ACM). https://doi.org/10.1145/3106668.3106672

Yamakata, Y., Imahori, S., Maeta, H., & Mori, S. (2016). A method for extracting major workflow composed of ingredients, tools, and actions from cooking procedural text. In *Proceedings of the 2016 IEEE International Conference on Multimedia & Expo Workshops* (pp. 1–6). IEEE Computer Society. https://doi.org/10.1109/ICMEW.2016.7574705

Yamakata, Y., Imahori, S., Sugiyama, Y., Mori, S., & Tanaka, K. (2013). Feature Extraction and Summarization of Recipes Using Flow Graph. In A. Jatowt, E. Lim, Y. Ding, A. Miura, T. Tezuka, G. Dias, K. Tanaka, A. J. Flanagin, & B. T. Dai (Eds.), *Proceedings of the 5th International Conference on Social Informatics*

(pp. 241–254). Springer International Publishing. https://doi.org/10.1007/978-3-319-03260-3_21

Yamakata, Y., Mori, S., & Carroll, J. A. (2020). English Recipe Flow Graph Corpus. In N. Calzolari, F. Béchet, P. Blache, K. Choukri, C. Cieri, T. Declerck, S. Goggi, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the 12th International Conference on Language Resources and Evaluation* (pp. 5187–5194). European Language Resources Association (ELRA).

Yasukawa, M., Diaz, F., Druck, G., & Tsukada, N. (2014). Overview of the NTCIR-11 Cooking Recipe Search Task. In N. Kando, H. Joho, & K. Kishida (Eds.), *Proceedings of the 11th NTCIR Conference* (pp. 483–496). National Institute of Informatics (NII).

Yoneda, Y., & Nadamoto, A. (2018). Knack for Cooking Extraction from User Generated Recipe Sites. In M. Indrawan-Santiago, E. Pardede, I. L. Salvadori, M. Steinbauer, I. Khalil, & G. Anderst-Kotsis (Eds.), *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services* (pp. 134–137). Association for Computing Machinery (ACM). https://doi.org/10.1145/3282373.3282404

Zhang, D., Mishra, S., Brynjolfsson, E., Etchemendy, J., Ganguli, D., Grosz, B., Lyons, T., Manyika, J., Niebles, J. C., & Sellitto, M. (2021). The AI Index 2021 Annual Report. *arXiv preprint arXiv:2103.06312*. https://doi.org/10.48550/arXiv.2103.06312

Zhang, Y., Yamakata, Y., & Tajima, K. (2022). MIAIS: A Multimedia Recipe Dataset with Ingredient Annotation at Each Instructional Step. In Y. Yamakata, A. Hashimoto, & J. Chen (Eds.), *Proceedings of the 1st International Workshop on Multimedia for Cooking, Eating, and related APPlications* (pp. 49–52). Association for Computing Machinery (ACM). https://doi.org/10.1145/3552485.3554938

Zhao, J., Wang, T., Yatskar, M., Ordonez, V., & Chang, K.-W. (2018). Gender Bias in Coreference Resolution: Evaluation and Debiasing Methods. In M. A. Walker, H. Ji, & A. Stent (Eds.), *Proceedings of the 2018 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 15–20). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/N18-2003

Appendices

Appendix A

Recipe-Related Benchmarks

In this appendix we will give an overview of all recipe-related benchmarks that we found during our background research as a useful summary that can be used in further research. Since this thesis focuses on research published in English, only benchmarks related to such research have been included.

Recipe Corpus	**Additional Contents**
CURD dataset (Tasse & Smith, 2008)	MILK representations
SIMMR Recipe Dataset (Jermsurawong & Habash, 2015)	SIMMR dependency trees
RecipeQA (Yagcioglu et al., 2018)	Question-answer pairs
RecipeScape (Chang et al., 2018)	Instruction Analysis Tool
Recipe1M+ (Marin et al., 2021)	Food images
Storyboarding dataset (K. Chandu et al., 2019)	Cooking images
English r-FG Corpus (Yamakata et al., 2020)	Recipe flow graphs
Microsoft Research Multimodal Aligned Recipe Corpus (Lin et al., 2020)	Timestamped YouTube URLs
MM-ReS dataset (Pan et al., 2020b)	Cooking images and workflows
RecipeDB (Batra et al., 2020)	Nutritional and cultural relations
ARA corpus (Donatelli et al., 2021)	Action graphs
Cooking Program dataset (Donatelli et al., 2021)	Program representations

Table A.1: An overview of English recipe-related corpora.

Recipe Corpus	**Additional Contents**
r-FG Corpus (Mori, Maeta, Yamakata, & Sasada, 2014)	Recipe flow graphs
KUSK (Hashimoto et al., 2014)	Sensor data from cooking process
Cookpad Recipe Dataset (Harashima et al., 2016)	Recipe combinations and reviews
Cookpad Image Dataset (Harashima et al., 2017)	Cooking images
r-FG BB dataset (Nishimura et al., 2020)	Cooking images and bounding boxes
CPC (Harashima & Hiramatsu, 2020)	Linguistic annotations
MIAIS (Y. Zhang et al., 2022)	Cooking images and step ingredients

Table A.2: An overview of Japanese recipe-related corpora that have accompanying English papers.

Appendix B

MUHAI Cooking Language Primitives

The primitives that are currently supported in MUHAI Cooking Language, their intended meaning and simulator-specific implementation choices will be listed and described alphabetically in the following paragraphs, with the number after / indicating the number of arguments the predicate takes. Similar information can also be found in the documentation accompanying our benchmark[1].

bake/9

- Arguments:
  ?baked-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-bake*, *?oven*, *?time-value*, *?time-unit*, *?temperature-value*, *?temperature-unit*
- Intended Meaning:
  Obtain *?baked-thing* by baking *?thing-to-bake* in *?oven* at the temperature specified by *?temperature-value* and *?temperature-unit* for the duration specified by *?time-value* and *?time-unit*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?oven* defaults to the closest unused oven in the kitchen
  - *?temperature-value* and *?temperature-unit* default to the temperature of *?oven* (only possible in case *?oven* is specified)
- Constant Arguments:
  - *?time-value* and *?temperature-value* can be numerical values
  - *?time-unit* can be *hour* or *minute*
  - *?temperature-unit* can be *degrees-celsius*

[1] https://ehai.ai.vub.ac.be/recipe-execution-benchmark

boil/8

- Arguments:
  ?boiled-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-boil*, *?stove*, *?heating-setting*, *?time-value*, *?time-unit*
- Intended Meaning:
  Obtain *?boiled-thing* by boiling *?thing-to-boil* on the *?stove* at the heating setting specified by *?heating-setting* for the duration specified by *?time-value* and *?time-unit*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?stove* defaults to the closest unused stove in the kitchen
  - *?heating-setting* defaults to *medium-heat*
  - *?time-value* and *?time-unit* default to 30 minutes
- Constant Arguments:
  - *?heating-setting* can be *low-heat*, *medium-heat*, *medium-high-heat*, *high-heat*
  - *?time-value* and *?temperature-value* can be numerical values
  - *?time-unit* can be *hour* or *minute*

beat/5

- Arguments:
  ?beaten-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-beat*, *?beating-tool*
- Intended Meaning:
  Obtain *?beaten-thing* by beating *?thing-to-beat* using *?beating-tool*. Beating can be seen as a more intense form of mixing which adds some air bubbles during the combination process. This form of mixing also leads to a homogeneous result as individual components are not kept intact. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?beating-tool* defaults to the closest unused whisk in the kitchen

bring-to-temperature/6

- Arguments:
  ?thing-at-desired-temperature, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-bring-to-temperature*, *?temperature-value*, *?temperature-unit*

- Intended Meaning:
  Obtain *?thing-at-desired-temperature* at the temperature specified by *?temperature-value* and *?temperature-unit* by waiting for *?thing-to-bring-to-temperature* to cool off or warm up by advancing towards the ambient temperature. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:

  – *?temperature-value* and *?temperature-unit* default to the current room temperature of the kitchen, which is around 18 °C.

- Constant Arguments:

  – *?temperature-value* can be a numerical value

  – *?temperature-unit* can be *degrees-celsius*

cover/5

- Arguments:
  ?covered-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-cover*, *?cover*

- Intended Meaning:
  Obtain *?covered-thing* by covering *?thing-to-cover* with the specified *?cover*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:

  – *?cover* defaults to an appropriate cover for the given *?thing-to-cover*, i.e., a *bowl-lid* for a *bowl*, a *jar-lid* for a *jar* or *plastic-wrap* for anything else.

cut/6

- Arguments:
  ?cut-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-cut*, *?cutting-pattern*, *?cutting-tool*

- Intended Meaning:
  Obtain *?cut-thing* by using *?cutting-tool* to cut *?thing-to-cut* according to the specified *?cutting-pattern*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:

  – *?cutting-tool* defaults to the closest unused knife in the kitchen

- Constant Arguments:

- *?cutting-pattern* can be *chopped*, *finely-chopped*, *slices*, *fine-slices*, *squares*, *two-cm-cubes*, *halved*, *shredded*, *minced*, *diced*

crack/5

- Arguments:
  ?container-with-whole-eggs, *?kitchen-state-out*, *?kitchen-state-in*, *?eggs-to-crack*, *?target-container-for-whole-eggs*
- Intended Meaning:
  Obtain *?container-with-whole-eggs* by cracking *?eggs-to-crack*, i.e., removing the egg shell from *?eggs-to-crack*, and dropping the egg contents in the container specified by *?target-container-for-whole-eggs*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?target-container-for-whole-eggs* defaults to the closest unused medium bowl in the kitchen

dip/5

- Arguments:
  ?dipped-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-dip*, *?dip*
- Intended Meaning:
  Obtain *?dipped-thing* by dipping *?thing-to-dip* into *?dip*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

drain/6

- Arguments:
  ?drained-thing, *?remaining-liquid*, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-drain*, *?draining-tool*
- Intended Meaning:
  Obtain *?drained-thing* by draining *?thing-to-drain* using *?draining-tool* leaving the remaining liquid in *?remaining-liquid*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?draining-tool* defaults to the closest unused colander in the kitchen

fetch/5

- Arguments:
  ?fetched-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-fetch*, *?quantity-to-fetch*

- Intended Meaning:
  Obtain *?fetched-thing* by locating one or more *?thing-to-fetch* objects in the kitchen and bringing it to a common work area such as a kitchen countertop. The exact number of objects to fetch is specified by *?quantity-to-fetch*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Constant Arguments:

  - *?quantity-to-fetch* can be a numerical value
  - *?thing-to-fetch* can be any transferable container or cooking utensil available in the kitchen environment. These can currently all be found in the *kitchen-cabinet* of the initial kitchen state shown in Figure C.1.

fetch-and-proportion/7

- Arguments:
  ?fetched-and-proportioned-ingredient, *?kitchen-state-out*, *?kitchen-state-in*, *?target-container-for-proportioned-ingredient*, *?mach-and-proportion*, *?proportion-value*, *?proportion-unit*

- Intended Meaning:
  Obtain an amount of the food product *?fetched-and-proportioned-ingredient* by fetching an *?ingredient-to-fetch-and-proportion*, taking a portion from it specified by *?proportion-value* and *?proportion-unit* and placing this portion inside the container specified by *?target-container-for-proportioned-ingredient*. Ingredient leftovers are returned to their original location. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Constant Arguments:

  - *?ingredient-to-fetch-and-proportion* can be any ingredient that is mentioned in the ingredient list of a supported recipe. Ingredients should be specified by replacing all spaces in an ingredient name with the minus sign (-), e.g., 'ground black pepper' would become *ground-black-pepper*. A list of all ingredients can be found by combining the contents of the *fridge*, *freezer* and *pantry* of the initial kitchen state shown in Figure C.1.
  - *?proportion-value* can be a numerical value
  - *?proportion-unit* can be *piece*, *g*, *teaspoon*, *tablespoon*, *l* or *ml*

flatten/5

- Arguments:
  ?flattened-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-flatten*, *?flattening-tool*
- Intended Meaning:
  Obtain *?flattened-thing* by flattening *?thing-to-flatten* using *?flattening-tool*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?flattening-tool* defaults to the closest unused rolling pin in the kitchen

flour/5

- Arguments:
  ?floured-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-flour*, *?flour*
- Intended Meaning:
  Obtain *?floured-thing* by flouring *?thing-to-flour* with *?flour*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?flour* defaults to 10 grams of all-purpose flour taken from the closest container with all-purpose flour

fry/8

- Arguments:
  ?fried-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-fry*, *?stove*, *?heating-setting*, *?time-value*, *?time-unit*
- Intended Meaning:
  Obtain *?fried-thing* by frying *?thing-to-fry* on the *?stove* at the heating setting specified by *?heating-setting* for the duration specified by *?time-value* and *?time-unit*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?stove* defaults to the closest unused stove in the kitchen
  - *?heating-setting* defaults to *medium-heat*
  - *?time-value* and *?time-unit* default to 30 minutes

- Constant Arguments:
  - *?heating-setting* can be *low-heat*, *medium-heat*, *medium-high-heat*, *high-heat*
  - *?time-value* and *?temperature-value* can be numerical values
  - *?time-unit* can be *hour* or *minute*

get-kitchen/1

- Arguments:
  ?initial-kitchen-state
- Intended Meaning:
  Obtain the initial state of the kitchen *?initial-kitchen-state*. This is expected to provide access to an environment model of the kitchen to provide contextual information needed for executing a recipe.
- Default Values:
  - *?initial-kitchen-state* defaults to the initial kitchen state in which a recipe will be executed. This argument is expected to be left to its default value in which case this primitive functions as a 'getter'.

grease/5

- Arguments:
  ?greased-thing, *?kitchen-state-out*, *?kitchen-state-in*,
  ?thing-to-grease, *?thing-to-grease*, *?grease*
- Intended Meaning:
  Obtain *?greased-thing* by greasing *?thing-to-grease* with *?grease*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?grease* defaults to 10 grams of butter taken from the closest container with butter

grind/5

- Arguments:
  ?ground-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-grind*, *?grinding-tool*
- Intended Meaning:
  Obtain *?ground-thing* by grinding *?thing-to-grind* using *?grinding-tool*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:
  - *?grinding-tool* defaults to the closest unused food-processor in the kitchen

leave-for-time/6

- Arguments:
  ?cooled-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-cool*, *?time-value*, *?time-unit*
- Intended Meaning:
  Obtain *?cooled-thing* by waiting for the duration specified by *?time-value* and *?time-unit* to let *?thing-to-cool* cool off towards the ambient temperature. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Constant Arguments:
  - *?time-value* and *?temperature-value* can be numerical values
  - *?time-unit* can be *hour* or *minute*

line/5

- Arguments:
  ?lined-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-line*, *?lining*
- Intended Meaning:
  Obtain *?lined-thing* by lining *?thing-to-line* with *?lining*, e.g., lining a baking tray with some baking paper or lining muffin tins with paper baking cups. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?lining* defaults to the closest unused sheet of baking paper
- Constant Arguments:
  - *?lining* can be *baking-paper* or *paper-baking-cups*
  - *?thing-to-line* can be *baking-tray*, *cookie-sheet*, *pan* or *muffin-tins*

mash/5

- Arguments:
  ?mashed-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-mash*, *?mashing-tool*

- Intended Meaning:
  Obtain *?mashed-thing* by mashing up *?thing-to-mash* using *?mashing-tool*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?mashing-tool* defaults to the closest unused fork in the kitchen

melt/5

- Arguments:
  ?melted-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-melt*, *?melting-tool*
- Intended Meaning:
  Obtain *?melted-thing* by melting *?thing-to-melt* using *?melting-tool*. This melting tool could be any kind of heating appliance in the kitchen, ranging from a pan on the stove to a microwave. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?melting-tool* defaults to the closest unused microwave in the kitchen

mingle/5

- Arguments:
  ?mingled-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-mingle*, *?mingling-tool*
- Intended Meaning:
  Obtain *?mingled-thing* by mingling *?thing-to-mingle* using *?mingling-tool*. Mingling can be seen as a softer form of mixing in which the individual components are still kept intact during the combination process. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?mingling-tool* defaults to the closest unused wooden spoon in the kitchen

mix/5

- Arguments:
  ?mixed-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-mix*, *?mixing-tool*

- Intended Meaning:
  Obtain *?mixed-thing* by mixing *?thing-to-mix* using *?mixing-tool*. Mixing can be seen as a form of mixing that is intense enough to achieve a homogeneous mixture without being so intense that air bubbles are added during the mixing process. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:

  – *?mixing-tool* defaults to the closest unused whisk in the kitchen

peel/6

- Arguments:
  ?peeled-thing, *?peel*, *?kitchen-state-out* *?kitchen-state-in*, *?thing-to-peel*, *?peeling-tool*

- Intended Meaning:
  Obtain *?peeled-thing* and its *?peel* by peeling *?thing-to-peel* using *?peeling-tool*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:

  – *?peeling-tool* defaults to the closest unused knife in the kitchen

portion-and-arrange/8

- Arguments:
  ?portions, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-portion*, *?portion-size-value*, *?portion-size-unit*, *?placement-pattern*, *?container-for-portions*

- Intended Meaning:
  Obtain *?portions* by portioning *?thing-to-portion* into portions that each have a size specified by *?portion-size-value* and *?portion-size-unit*. These portions are placed onto the container *?container-for-portions* following the *?placement-pattern*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:

  – *?placement-pattern* defaults to a pattern in which all portions are evenly spread out over the container

  – *?container-for-portions* defaults to the countertop of the kitchen

  – *?portion-size-value* and *?portion-size-unit* default to portion sizes that cause an equal division over the available tins (only possible in case *?container-for-portions* are muffin tins)

- Constant Arguments:
  - *?placement-pattern* can be *side-to-side*, *evenly-spread*, *5-cm-apart*

preheat-oven/6

- Arguments:
  ?preheated-oven, *?kitchen-state-out*, *?kitchen-state-in*, *?oven*, *?temperature-value*, *?temperature-unit*
- Intended Meaning:
  Obtain *?preheated-oven* by changing the settings of the *?oven* to reach the temperature specified by *?temperature-value* and *?temperature-unit*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?oven* defaults to the closest unused oven in the kitchen
- Constant Arguments:
  - *?temperature-value* can be a numerical value
  - *?temperature-unit* can be *degrees-celsius*

refrigerate/7

- Arguments:
  ?refrigerated-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-refrigerate*, *?refrigerator*, *?time-value*, *?time-unit*
- Intended Meaning:
  Obtain *?refrigerated-thing* by putting *?thing-to-refrigerate* inside *?refrigerator* for the duration specified by *?time-value* and *?time-unit*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?refrigerator* defaults to the closest unused fridge in the kitchen
  - *?time-value* and *?time-unit* default to one hour
- Constant Arguments:
  - *?time-value* can be a numerical value
  - *?time-unit* can be *minute* or *hour*

seed/6

- Arguments:
  ?seeded-thing, *?seed*, *?kitchen-state-out* *?kitchen-state-in*, *?thing-to-seed*, *?seeded-tool*
- Intended Meaning:
  Obtain *?seeded-thing* and its *?seed* by seeding *?thing-to-seed* using *?seeding-tool*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?seeding-tool* defaults to the closest unused knife in the kitchen

separate-eggs/8

- Arguments:
  ?egg-yolks, *egg-whites*, *?kitchen-state-out*, *?kitchen-state-in*, *?eggs*, *?container-for-yolks*, *?container-for-whites*, *?egg-separator*
- Intended Meaning:
  Obtain *?egg-yolks* and *egg-whites* by using an *?egg-separator* to separate separating the whole *?eggs* into the *?container-for-yolks* and *?container-for-whites* respectively. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?container-for-yolks* defaults to the closest unused stove in the kitchen
  - *?container-for-whites* defaults to the closest unused medium bowl in the kitchen (excluding the one found for *?container-for-yolks*)
  - *?egg-separator* defaults the closest unused egg separator in the kitchen

shake/4

- Arguments:
  ?shaken-thing, *?kitchen-state-out* *?kitchen-state-in*, *?thing-to-shake*
- Intended Meaning:
  Obtain *?shaken-thing* by shaking *?thing-to-shake* to mix its contents, which are generally liquids, until a homogeneous mixture is reached. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

shape/5

- Arguments:
  ?shaped-thing, *?kitchen-state-out* *?kitchen-state-in*, *?thing-to-shape*, *?shape*
- Intended Meaning:
  Obtain *?shaped-thing* by shaping *?thing-to-shape* into the shape specified by *?shape*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Constant Arguments:
  - *?shape* can be *ball-shape* or *crescent-shape*

sift/6

- Arguments:
  ?sifted-thing, *?kitchen-state-out* *?kitchen-state-in*, *?container-to-sift-into*, *?thing-to-sift*, *?sift*
- Intended Meaning:
  Obtain *?sifted-thing* by using *?sift* to sift *?thing-to-sift* into the container specified by *?container-to-sift-into*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?container-to-sift-into* defaults to the closest unused large bowl in the kitchen
  - *?sift* defaults to the closest unused sift in the kitchen

spread/6

- Arguments:
  ?thing-with-spread-on, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-spread-on*, *?thing-to-spread*, *?spreading-tool*
- Intended Meaning:
  Obtain *?thing-with-spread* by spreading *?thing-to-spread* on *?thing-to-spread-on* using *spreading-tool*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?spreading-tool* defaults to the closest unused spatula in the kitchen

sprinkle/5

- Arguments:
  ?thing-with-sprinkles-on, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-sprinkle-on*, *?sprinkles*

- Intended Meaning:
  Obtain *?thing-with-sprinkles-on* by sprinkling *?sprinkles* onto *?thing-to-sprinkle-on*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

transfer-contents/8

- Arguments:
  ?container-with-transferred-contents, *?container-with-rest-of-contents*, *?kitchen-state-out*, *?kitchen-state-in*, *?container-to-transfer-contents-to*, *?container-with-contents-to-transfer*, *?value-of-transfer-amount*, *?unit-of-transfer-amount*

- Intended Meaning:
  Obtain *?container-with-transferred-contents* and *?container-with-rest-of-contents* by transferring an amount (specified by *?value-of-transfer-amount* and *?unit-of-transfer-amount*) of the container *?container-with-contents-to-transfer*'s contents into *?container-to-transfer-contents-to* leaving the remaining contents in *?container-with-rest-of-contents*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:
  - *?container-to-transfer-contents-to* defaults to the closest unused large bowl in the kitchen
  - *value-of-transfer-amount* and *unit-of-transfer-amount* default to an amount for which all contents are transferred, effectively emptying the original container

- Constant Arguments:
  - *?value-of-transfer-amount* can be a numerical value
  - *?unit-of-transfer-amount* can be *piece*, *g*, *teaspoon*, *tablespoon*, *ml* or *percent*

transfer-items/6

- Arguments:
  ?transferred-items, *?kitchen-state-out*, *?kitchen-state-in*, *?items-to-transfer*, *?placement-pattern*, *?destination*

- Intended Meaning:
  Obtain *?transferred-items* by carefully transferring all items from *?items-to-transfer* to *?destination* and placing them there according to the pattern specified by *?placement-pattern*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:

  – *?placement-pattern* defaults to a pattern in which the available location is filled up from side to side by creating rows of items one at a time in which items are placed next to each other.

uncover/5

- Arguments:
  ?uncovered-thing, *?cover* *?kitchen-state-out*, *?kitchen-state-in*, *?covered-thing*

- Intended Meaning:
  Obtain *?uncovered-thing* and its prior *?cover* by removing the cover from *?covered-thing*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

wash/4

- Arguments:
  ?washed-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-wash*

- Intended Meaning:
  Obtain *?washed-thing* by rinsing off or washing *?thing-to-wash* with water. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

Appendix C

Initial Kitchen State

In this appendix we will describe the composition of the initial kitchen state from which recipe execution will be started in the simulator for any of the available test recipes. The initial kitchen state provides the initial context to start from and gets loaded in by the *get-kitchen* operation, which should be present in every semantic network.

The contextual information provided by knowing the initial kitchen state could influence the actual interpretation of recipe texts, since it could alter what is possible and thus what the semantic network should look like. Therefore, the composition of the initial kitchen state should be taken into account during model development and evaluation.

There is currently only one initial kitchen state made available in our benchmark which is a very full kitchen containing all kitchen commodities, tools, appliances and ingredients that could be needed for executing any of the recipes in our benchmark. All ingredients are made available in medium bowls that could be located in the fridge, freezer or pantry based on the ingredient's storage requirements.

In Figure C.1 we give a detailed overview of the supported initial kitchen state, with more information about initial kitchen states in general being available in the documentation accompanying our benchmark[1].

[1] https://ehai.ai.vub.ac.be/recipe-execution-benchmark

kitchen *(temperature: 18 °C)*

counter-top

oven | **stove** | **microwave**

freezer *(temperature: -18 °C)*

- frozen-corn *(500 g)*

fridge *(temperature: 5 °C)*

- apple *(6 pcs)*
- avocado *(6 pcs)*
- banana *(6 pcs)*
- black-bean *(500 g)*
- black-olive *(100 g)*
- butter *(500 g)*
- broccoli *(500 g)*
- carrot *(12 pcs)*
- celery *(6 pcs)*
- cherry-tomato *(500 g)*
- cooked-bacon *(500 g)*
- cooked-chicken *(500 g)*
- corn *(500 g)*
- cranberry *(500 g)*
- cream-cheese *(500 g)*
- crushed-pineapple *(500 g)*
- cucumber *(10 pcs)*
- salted-butter *(500 g)*
- egg *(12 pcs)*
- egg-white *(500 g)*
- feta-cheese *(500 g)*
- fresh-basil *(500 g)*
- fresh-cilantro *(200 g)*
- fresh-oregano *(50 g)*
- grated-horesradish *(200 g)*
- grated-mozzarella *(500 g)*
- green-cabbage *(2 pcs)*
- green-chili-pepper *(5 pcs)*
- hard-boiled-egg *(12 pcs)*
- heavy-cream *(500 g)*
- jalapeno *(5 pcs)*
- lemon-juice *(500 ml)*
- lime-juice *(500 ml)*
- mango *(5 pcs)*
- mixed-greens *(500 g)*
- radish *(10 pcs)*
- red-bell-pepper *(10 pcs)*
- red-chili-pepper *(5 pcs)*
- romaine-lettuce *(2 pcs)*
- salted-butter *(500 g)*
- shredded-coconut *(500 g)*
- tomato *(12 pcs)*
- water *(1 l)*
- yellow-mustard *(500 ml)*

pantry

- almond *(500 g)*
- almond-extract *(100 g)*
- almond-flakes *(250 g)*
- almond-flour *(1 kg)*
- all-purpose-flour *(1 kg)*
- baking-powder *(250 g)*
- baking-soda *(50 g)*
- bisquick-baking-mix *(300 g)*
- brown-sugar *(1 kg)*
- caster-sugar *(1 kg)*
- celery-seed *(200 g)*
- cider-vinegar *(500 ml)*
- coarse-salt *(500 g)*
- cocoa-powder *(500 g)*
- coconut-oil *(500 ml)*
- corn-flakes *(500 g)*
- devils-food-cake-mix *(600 g)*
- dried-dill-weed *(500 g)*
- extra-virgin-olive-oil *(500 ml)*
- garlic *(5 pcs)*
- garlic-powder *(500 g)*
- green-onion *(10 pcs)*
- ground-allspice *(50 g)*
- ground-black-pepper *(500 g)*
- ground-cinnamon *(50 g)*
- ground-cloves *(50 g)*
- ground-cumin *(500 g)*
- ground-ginger *(50 g)*
- ground-nutmeg *(50 g)*
- marshmallow *(500 g)*
- molasses *(900 g)*
- mustard-seed *(200 g)*
- oats *(500 g)*
- onion *(10 pcs)*
- paprika-powder *(300 g)*
- potato *(12 pcs)*
- powdered-white-sugar *(500 g)*
- red-onion *(5 pcs)*
- red-pepper-flakes *(50 g)*
- red-wine-vinegar *(500 ml)*
- salt *(500 g)*
- self-rising-flour *(1 kg)*
- shallot *(5 pcs)*
- sweet-potato *(500 g)*
- tabasco *(250 ml)*
- turmeric-powder *(200 g)*
- vanilla *(500 g)*
- vanilla-extract *(100 g)*
- vegetable-oil *(200 g)*
- walnut *(500 g)*
- white-bread-slice *(12 pcs)*
- white-sugar *(1 kg)*
- white-vinegar *(500 g)*
- whole-wheat-flour *(1 kg)*

kitchen-cabinet

- baking-paper *(3)*
- baking-tray *(1)*
- bread-knife *(3)*
- colander *(3)*
- cookie-sheet *(1)*
- cooking-pot *(3)*
- egg-separator *(3)*
- food-processor *(1)*
- fork *(9)*
- jar *(3)*
- jar-lid *(3)*
- knife *(9)*
- large-bowl *(9)*
- large-bowl-lid *(3)*
- medium-bowl *(9)*
- medium-bowl-lid *(3)*
- mixer *(3)*
- muffin-tins *(1)*
- paper-baking-cup *(15)*
- plastic-wrap *(1)*
- rolling-pin *(3)*
- sift *(3)*
- small-bowl *(9)*
- small-bowl-lid *(3)*
- spatula *(3)*
- table-spoon *(9)*
- tea-spoon *(9)*
- whisk *(9)*
- wire-rack *(1)*
- wooden-spoon *(9)*

Figure C.1: Visualization of the simulator's initial kitchen state.

Appendix D

Pseudo-Algorithms for the Dish Approximation Score

To give a more formal explanation of the 'dish unfolding' and 'dish approximation score' algorithms, we will describe both of them in pseudocode. Section D.1 describes how to compute the dish approximation score, while Section D.2 describes how to unfold a dish. It should be noted that certain optimizations and edge cases have not been included in this pseudocode for the sake of clarity. However, the complete code has been made publicly available as part of the Babel package[1].

D.1 Pseudo-Algorithm for Dish Approximation Score

DishApproximationScore(GoldStandardDish, PredictedOutputs)

parameters
GoldStandardDish - the main output food product of the gold standard solution
PredictedOutputs - the set of final output food products from the prediction

implementation:

```
MaxScore ← 0
for every Output in PredictedOutputs do
    ContainerScore ← 0
    if SameLocation(Output, GoldStandardDish) then
        IncreaseScore(ContainerScore)
    else
        DecreaseScore(ContainerScore)
    end if
    for every Property in PropsOf(ContainerOf(GoldStandardDish)) do
        if hasProperty(ContainerOf(Output), Property) then
            IncreaseScore(ContainerScore)
```

[1] https://ehai.ai.vub.ac.be/recipe-execution-benchmark

```
        else
            DecreaseScore(ContainerScore)
        end if
    end for

    PredIngredients ← Unfold(Contents(Output), None)
    GSIngredients ← Unfold(Contents(GoldStandardDish), None)
    ContentsScores ← ∅
    for every GSIngredient in GSIngredients do
        MaxBaseScore ← 0
        MaxBaseIngredient ← ∅
        for every PredIngredient in PredIngredients do
            IngScore ← PropertyOverlap(PredIngredient, GSIngredient)
            SeqScore ← HierarchyOverlap(PredIngredient, GSIngredient)
            BaseScore ← 0.60 × IngScore + 0.40 × SeqScore
            if BaseScore > MaxBaseScore then
                MaxBaseScore ← BaseScore
                MaxBaseIngredient ← PredIngredient
            end if
        end for
        if MaxIngScore > 0 then
            GSIngredients ← GSIngredients \ {GSIngredient}
            PredIngredients ← PredIngredients \ {MaxBaseIngredient}
        end if
        ContentsScores ← ContentsScores ∪ {MaxBaseScore}
    end for
    for Length(GSIngredients) + Length(PredIngredients) times do
        ContentsScores ← ContentsScores ∪ {0}
    end for

    CurrentScore ← 0.02 × ContainerScore + 0.98 × AVG(ContentsScores)
    if CurrentScore > MaxScore then
        MaxScore ← CurrentScore
    end if
end for
return MaxScore
```

D.2 Pseudo-Algorithms for Dish Unfolding

Unfold(Dish)

parameters:
Dish - the dish whose contents should be unfolded into base ingredients

implementation:

$Unfolded \leftarrow \emptyset$
for every *Portion* in $Contents(Dish)$ **do**
  $Unfolded \leftarrow Unfolded \cup UnfoldIngredient(Portion, [])$
end for

$Merged \leftarrow \emptyset$
for every $BaseIng1$ in $Unfolded$ **do**
  for every $BaseIng2$ in $Unfolded \setminus \{BaseIng1\}$ **do**
    if $Similar(BaseIng1,\ BaseIng2)$ **then**
      $Amount(BaseIng1) \leftarrow Amount(BaseIng1) + Amount(BaseIng2)$
      $Unfolded \leftarrow Unfolded \setminus \{BaseIng2\}$
    end if
  end for
  $Merged \leftarrow Merged \cup \{BaseIng1\}$
  $Unfolded \leftarrow Unfolded \setminus \{BaseIng1\}$
end for

return *Merged*

UnfoldIngredient(Ingredients, Hierarchy)

parameters:
Ingredient - a single ingredient which might need additional unfolding (in case it is an aggregate food product)
Hierarchy - an ordered list of aggregate food products in which *Ingredient* is used

implementation:

if *Ingredient* is an aggregate food product **then**
  return $UnfoldIngredient(Ingredient,\ [Ingredient]\ +\ Hierarchy)$
else
  return $(Ingredient, Hierarchy)$
end if

Appendix E

Smatch Conversion Example

To give better insight into how a semantic network is converted into a conjunction of logical propositions that can be used for Smatch score computations, we will give a brief example of such a conversion for the following semantic network:

```
(get-kitchen ?ks-in)
(fetch-and-proportion ?proportioned-butter ?ks-out ?ks-in
    ?target-container butter 230 g)
```

As mentioned in section 3.4.3, there are three steps needed for conversion. These are

1. mapping each encountered primitive and variable to an instance;
2. creating an argument relation between each primitive name and its variable arguments;
3. creating an argument attribute for the primitive name with a fixed value every time a constant argument is encountered.

Step 1 There are two primitives and only four different variables, since *?ks-in* is a shared argument. This leads to the following conjunction of triples

$$
\begin{aligned}
&instance(a_0,\ get-kitchen) \wedge \\
&instance(a_1,\ var) \wedge \\
&instance(a_2,\ fetch-and-proportion) \wedge \\
&instance(a_3,\ var) \wedge \\
&instance(a_4,\ var) \wedge \\
&instance(a_5,\ var)
\end{aligned}
$$

In these triples, instances a_1, a_3, a_4 and a_5 respectively represent *?ks-in*, *?proportioned-butter*, *?ks-out* and *?target-container*.

Step 2 The primitive *get-kitchen* has one variable argument, namely *?ks-in* and the primitive *fetch-and-proportion* has four variable arguments namely *?proportioned-butter*, *?ks-out*, *?ks-in* and *?target-container*. Argument positions should be taken into account as well, which we do by adding an index to each relation name. $ARG2(a_2,\ a_1)$ for example indicates a_1 is the third argument in the primitive instance represented by a_2. Adding this type of argument relations leads to the following additional conjunction of triples

$$
\begin{aligned}
&ARG0(a_0,\ a_1) \wedge \\
&ARG0(a_2,\ a_3) \wedge \\
&ARG1(a_2,\ a_4) \wedge \\
&ARG2(a_2,\ a_1) \wedge \\
&ARG3(a_2,\ a_5)
\end{aligned}
$$

Step 3 The primitive *get-kitchen* has no constant arguments, but the primitive *fetch-and-proportion* has three of them which are *butter*, *230* and *g*. For the creation of attributes argument positions are taken into account by adding an index to each attribute name, similar to what was done in Step 2. $ATTR4(a_2,\ butter)$ for example indicates the fifth argument of a_2 has the fixed value *butter*. Adding this type of argument relations leads to the following additional conjunction of triples

$$
\begin{aligned}
&ATTR4(a_2,\ butter) \wedge \\
&ATTR5(a_2,\ 230) \wedge \\
&ATTR6(a_2,\ g)
\end{aligned}
$$

Steps 1-3 If we combine all three steps together, we get the following conjunction of logical propositions that is usable by Smatch

$$
\begin{aligned}
&instance(a_0,\ get-kitchen) \wedge \\
&instance(a_1,\ var) \wedge \\
&instance(a_2,\ fetch-and-proportion) \wedge \\
&instance(a_3,\ var) \wedge \\
&instance(a_4,\ var) \wedge \\
&instance(a_5,\ var) \wedge \\
&ARG0(a_0,\ a_1) \wedge \\
&ARG0(a_2,\ a_3) \wedge \\
&ARG1(a_2,\ a_4) \wedge \\
&ARG2(a_2,\ a_1) \wedge \\
&ARG3(a_2,\ a_5) \wedge \\
&ATTR4(a_2,\ butter) \wedge \\
&ATTR5(a_2,\ 230) \wedge \\
&ATTR6(a_2,\ g)
\end{aligned}
$$

This example also shows that a simple semantic network composed of only two primitives can already lead to a relatively long conjunction of 14 propositions. As we also explained in section 3.4.3, these type of metrics can therefore also become computationally costly for larger semantic networks.

Appendix F

Examples of Result Interpretations

In this appendix we will go through a number of example solutions aimed at providing a better understanding of how to interpret evaluation results. Moreover, these examples will further accentuate the importance of combining multiple metrics. The examples that are used are different possible solutions for the recipe 'Almond Crescent Cookies', which all have varying degrees of correctness. The actual solution files of these examples can be found with additional comments in the documentation accompanying our benchmark online[1].

F.1 Perfect Solution

The 'Almond Crescent Cookies' recipe requires an implicit step of warming up butter and fetching some unmentioned baking utensils. In a perfect solution tools are also reused as much as possible for maximum efficiency. The obtained scores for such a perfect solution are given in Table F.1.

Smatch	**Goal-Condition Success**	**Dish Score**	**Execution Time**
1.00	1.00	1.00	2600

Table F.1: An overview of the evaluation results for a perfect solution for the 'Almond Crescent Cookies' recipe.

Since all mandatory steps and no unneeded steps are included, the same implicit graph is obtained as in the gold standard solution. This is the only case in which we expect Smatch to detect full propositional overlap and a Smatch score of 1 is achieved. As the exact same implicit graph is obtained, goal-condition success and dish approximation score are also 1. The recipe execution time is the minimum that is possible and coincides with the gold standard recipe execution time, which is 2600 time steps.

[1] https://ehai.ai.vub.ac.be/recipe-execution-benchmark

F.2 Perfect Solution With Permuted Sequence

We should be able to randomly permute the sequence of primitive statements in a solution without changing evaluation results. The obtained scores for such a solution are given in Table F.2.

Smatch	Goal-Condition Success	Dish Score	Execution Time
1.00	1.00	1.00	2600

Table F.2: An overview of the evaluation results for a perfect solution for the 'Almond Crescent Cookies' recipe with the primitive statements themselves being randomly permuted in the solution file.

The sequence of the primitives in the file itself are unimportant, since all metrics are based on the implicit execution graph that is formed through argument sharing. This implicit graph is still the same as the implicit graph of the gold standard solution, meaning all results are still optimal.

F.3 Solution With Switched Operations

In contrast to only permuting statements in the file, switching the actual execution order of operations can have an influence on evaluation metrics. In Table F.3 we show the scores obtained for a near-perfect solution in which some ingredients were added to the bowl in a different order than specified by the steps in the recipe.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.99	0.92	1.00	2600

Table F.3: An overview of the evaluation results for a near-perfect solution for the 'Almond Crescent Cookies' recipe in which some ingredients were added to the bowl in a different order than specified by the steps in the recipe.

Since operational steps have been switched in the implicit graph, there is no complete propositional overlap with the gold standard graph and thus the Smatch score is slightly lowered. Additionally, goal-condition success is lowered as well. If two goal-conditions are to have a bowl with 'ingredient1' and then to have a bowl with 'ingredient1' and 'ingredient2', then adding 'ingredient2' before 'ingredient1' leads to the first goal-condition never being reached. Nevertheless, just adding ingredients in a bowl in a different order before mixing them should not have an impact on the taste which is the reason the dish approximation score is still maximal. Furthermore, adding ingredients in a different order should not lead to a change in recipe execution time either which is the reason recipe execution time is still the same as for the perfect solution.

F.4 Inefficient Solution

The gold standard solution tries to be as efficient as possible which means it tries to maximize reuse of tools. A solution that is less efficient will therefore differ in certain evaluation scores compared to a more efficient solution. In Table F.4 we show the scores obtained for a solution that contains all required steps, but does not always reuse cooking tools when possible.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.93	1.00	1.00	2660

Table F.4: An overview of the evaluation results for an inefficient solution for the 'Almond Crescent Cookies' recipe that contains all required steps, but does not always reuse cooking tools when possible.

Since tools are not reused, this means that strictly speaking some unnecessary steps have been executed. Therefore, the implicit graph is different from the gold standard solution which results in a lower Smatch score. Furthermore, fetching new tools also increases the recipe execution time as tool reuse is more efficient.

Important to note here is that all gold standard goal-conditions are still reached and the final dish is completely correct, which means the goal-condition success and the dish approximation score are still maximal. We will still successfully execute the recipe, but it will simply happen in an inefficient way. Therefore, from a simulation-based perspective adding the recipe execution time is actually quite informative in this particular case.

F.5 Solution With a Minor Step Missing

If required steps are missing this should have an effect on evaluation results. In Table F.5 we show the scores obtained for a solution in which the cooking agent failed to deduce the implicit step of letting the butter warm up before mixing it. This is a mistake in recipe execution, but should generally be only a minor inconvenience for the later mixing process.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.94	0.38	0.99	1980

Table F.5: An overview of the evaluation results for an imperfect solution for the 'Almond Crescent Cookies' recipe in which the cooking agent failed to deduce the implicit step of letting the butter warm up before mixing it.

Since a mandatory step is missing, the implicit graph is different from the gold standard solution which results in a lower Smatch score as there is no complete propositional overlap. Furthermore, warming up the butter was a required step and thus a goal-condition that is missed.

It is important to note here that this missed goal-condition occurs quite early in the process leading to many subsequent goal-condition failures and a low goal-condition success score overall. Every goal-condition that directly or indirectly requires this warmed up butter will now fail. The evaluation of goal-conditions is very strict, with a goal-condition either being completely fulfilled or being unsatisfied. This is a clear example of why the addition of the dish approximation score is useful. Not warming up the butter will not have a large influence on the taste of the final dish which is represented in a high dish approximation score.

A second important note here is that the recipe execution time is actually lower than the recipe execution time of a perfect solution. This is normal as we executed less steps and therefore finished quicker. Therefore, recipe execution time should generally not be interpreted in isolation as it is only informative in combination with the other metrics.

F.6 Partial Solution

The type and timing of steps that are missing will lead to a different effect on evaluation results. In Table F.6 we show the scores obtained for a solution in which the cooking agent failed to deduce fetching a baking tray and baking paper near the end of the recipe. This led it to also skip the later steps since they required those cooking utensils.

Smatch	**Goal-Condition Success**	**Dish Score**	**Execution Time**
0.84	0.77	0.82	1320

Table F.6: An overview of the evaluation results for an imperfect solution for the 'Almond Crescent Cookies' recipe in which the cooking agent failed to deduce fetching a baking tray and baking paper near the end of the recipe. This led it to also skip the later steps since they required those cooking utensils.

Since some mandatory steps are missing, the implicit graph is different from the gold standard solution which results in a lower Smatch score as there is no complete propositional overlap. Furthermore, all goal-conditions after fetching the baking tray are missing which leads to a significant lowering of goal-condition success as well. Since the mistake happens fairly late in the process, there are less goal-conditions impacted by the mistake compared to forgetting an operation early on, even if that early operation would have been less significant. Forgetting to bake cookies at the end will have a lower impact than forgetting to warm up butter at the start of cooking, since goal-condition success considers all properties to be equally important.

The dish approximation score has been lowered as well. It should be noted, however, that this lowering is mainly caused by the fact that an ingredient is missing due to the absence of sugar sprinkling which would have happened after baking. When it comes to taste, the impact of not baking the dough might be equally high in practice, but the dish score mostly biases correctness of ingredient composition. This seems defensible as it is easy for a human to still intervene and go from portioned dough to baked cookies,

while human intervention would not allow turning incorrectly made baked cookies into correctly made cookies.

Since the baking operation is the most time-consuming step in this recipe, recipe execution time is much lower than the perfect solution. This example thus demonstrates again that this metric is not very informative in absence of the other metrics.

F.7 Solution Using a Wrong Ingredient

Switching a base ingredient with another ingredient can lead to a significantly different dish and thus has an effect on evaluation results. In Table F.7 we show the scores obtained for a solution in which the cooking agent switched the base ingredient white sugar with cocoa powder.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.99	0.42	0.76	2600

Table F.7: An overview of the evaluation results for an imperfect solution for the 'Almond Crescent Cookies' recipe in which the cooking agent switched the base ingredient white sugar with cocoa powder.

Since a mandatory step has changed, the implicit graph is different from the gold standard solution which results in a lower Smatch score as there is no complete propositional overlap. It is important to note here that changing a base ingredient does not lead to a significant lowering of the Smatch score, although it can be expected to have a significant impact on the final dish. The achieved Smatch score is even comparable to a setting in which only a minor step is forgotten, such as letting butter warm up before adding it. This further highlights the importance of including other metrics than Smatch score.

Both goal-condition success and the dish approximation score have been heavily affected by the ingredient mistake. The missed goal-conditions of fetching and adding white sugar happened early on in the process, which led to many subsequent goal-condition failures and thus a low goal-condition success score overall. Every goal-condition directly or indirectly requiring the sugar ingredient has failed.

In addition, this particular mistake is also expected to have a big influence on the taste of the final dish. Therefore, the dish approximation score is substantially lowered as well, resulting in a score that is justifiably lower than a case in which a minor step such as warming up butter is forgotten.

However, recipe execution time has remained optimal as both ingredients should take equally long to be fetched and added to a bowl. This example thus demonstrates once more that recipe execution time is not informative in isolation.

F.8 Solution With an Additional Side Dish

Creating an additional side dish that is not included in the original recipe does not alter the main dish, but will lead to an inefficient use of time if the main objective is preparing the main dish. Therefore, it has an effect on evaluation results. In Table F.8 we show the scores obtained for a solution in which an additional chocolate side dip is prepared after the main dish is made.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.90	1.00	1.00	2740

Table F.8: An overview of the evaluation results for an imperfect solution for the 'Almond Crescent Cookies' recipe in which an additional unmentioned chocolate side dip is prepared after the main dish is made.

Since a surplus of steps is present due to the side dish preparation, the implicit graph is different from the gold standard solution which results in a lower Smatch score. Additionally, recipe execution time has increased due to the extra steps that have been executed.

However, both goal-condition success and the dish approximation score are still maximal. Even if more steps are executed than needed for the main dish, all required goal-conditions have been achieved leading to perfect goal-condition success. Furthermore, the main dish that we wanted to make is available in perfect condition at the end leading to a perfect dish approximation score.

It is important to note here that the dish approximation score is not based on the side dish, even though this is the last dish that was prepared during recipe execution. The dish approximation score will always be computed on the dish that maximally approximates the gold standard dish irrespective of the timing of its availability. Therefore, this is another case in which recipe execution time is useful to distinguish efficient from inefficient time use as both other simulation-based metrics do not detect this.

F.9 Solution With an Extended Main Dish

Preparing a dip that is not included in the original recipe might already be an inefficient use of time, but dipping the cookies from the main dish into this dip would also effectively alter the main dish. Therefore, adding such an operation will have a bigger impact on evaluation results. In Table F.9 we show the scores obtained for a solution in which an additional chocolate dip is prepared and the main dish's cookies are dipped into it at the end.

Since a surplus of steps is present due to the dip preparation and dipping operation, the implicit graph is different from the gold standard solution which results in a lower Smatch score. Additionally, recipe execution time has increased due to the extra steps that have been executed.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.92	1.00	0.87	2790

Table F.9: An overview of the evaluation results for an imperfect solution for the 'Almond Crescent Cookies' recipe in which an additional chocolate dip is prepared and the main dish's cookies are dipped into it at the end.

Furthermore, the dish approximation score has been affected as well. Dipping the main dish's cookies in chocolate alters their composition. Even though the main dish was available in perfect condition at some point during execution, the final dish that can be served is not perfectly comparable with the dish we aimed to create which leads to a lower dish approximation score.

However, it is important to note that goal-condition success is still maximal in this case. More steps are executed than needed to reach a state in which the main dish is available, but all required goal-conditions have still been achieved at some point which leads to perfect goal-condition success. Goal-condition success is thus independent of the fact that some goal-conditions might be undone at a later point by altering the main dish, as long as the goal-conditions have been reached once. This example thus demonstrates why combining multiple metrics can be important in order to obtain a reliable view on performance.

F.10 Solution Without Actual Cooking

In case recipe parsing fails almost completely, evaluation results should adequately demonstrate such a failure occurred. In Table F.10 we show the scores obtained for a solution in which no actual cooking takes place as only the fetch operations were recognized, namely fetching a baking tray and baking paper.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.12	0.08	0.00	60

Table F.10: An overview of the evaluation results for an imperfect solution for the 'Almond Crescent Cookies' recipe in which no actual cooking takes place as only the fetch operations were recognized, namely fetching a baking tray and baking paper.

Since many mandatory steps are missing, the implicit graph is very different from the standard solution which results in a low Smatch score. However, it should be noted that a score of 0.12 for an execution network that performs no actual cooking is still relatively high. Moreover, since fetching a baking tray and baking paper were actual goal-conditions, goal-condition success is also not 0 and even relatively high for a recipe execution network that performs no actual cooking. These results again highlight the importance of using a combination of metrics as they provide multiple perspectives on the same performance. The dish approximation score for example is effectively 0 which

clearly indicates no useful food product has been made.

Recipe execution time is very low as not many cooking operations have actually taken place. This emphasizes once more that recipe execution time is only informative in combination with the other metrics. Optimizing solely for recipe execution time could lead to results in which no cooking takes place.