EPFL

Thèse n° 10 872

Collaborative AI Agents in the Era of Large Language Models

Présentée le 2 mai 2025

Faculté informatique et communications
Laboratoire de science des données
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Martin JOSIFOSKI

Acceptée sur proposition du jury

Prof. V. Kuncak, président du jury
Prof. R. West, directeur de thèse
Prof. P. Minervini, rapporteur
Dr C. Musat, rapporteur
Prof. C. Gulcehre, rapporteur

École polytechnique fédérale de Lausanne

2025

To my family.

Acknowledgements

I want to thank my advisor, Robert West, who has profoundly influenced my journey over the last 7 years. Initially, the research assistant position he offered drew me to EPFL for my master's studies, and later, the intellectually stimulating environment he nurtured at the lab made pursuing a PhD a natural choice. I am deeply grateful for the freedom and guidance he provided, both directly and indirectly, which fostered in me the confidence to pursue ambitious ideas and confront my own ignorance. I would also like to thank Viktor Kuncak, Caglar Gulcehre, Pasquale Minervini, and Claudiu Musat, who kindly agreed to serve on my PhD committee.

I am very grateful to Maxime Peyrard, who has been an exceptional collaborator and friend through these years. His presence has been a key factor in my productivity and motivation throughout my PhD journey. The ideas in this thesis owe a great deal to our discussions, and I continue to learn from him about both work and life.

Throughout my PhD, I was fortunate to be part of many collaborations that introduced me to new ideas and exceptional people. At EPFL, I had the pleasure of working alongside talented colleagues and MS students, including Saibo Geng, Yifei Li, Frano Rajic, Debjit Paul, Jiheng Wei, Giovanni Monea, Julian Schnitzler, Yuxing Yao, Valentin Hartmann, Marija Sakota, Lars Klein, Akhil Arora, Venyamin Veselovsky, Tim Davidson and Manoel Ribeiro. I particularly want to highlight my collaboration with Nicolas Baldwin, who transitioned from an MS student to a colleague, a great researcher, and a friend, and made my PhD journey more exciting. Our long-term collaboration with Microsoft connected me with outstanding researchers like Vishrav Chaudhary, Barun Patra, Emre Kiciman, and Jason Eisner, while working with Mauro Staver and Dawn Song from Berkeley opened new research directions. For the final leg of my PhD, I am grateful to the Visual Intelligence and Learning Lab for providing a welcoming and stimulating work environment, and especially thank Andrei Atanov for being such a wonderful friend and one of my most enjoyable conversation partners.

At the end of my master's studies, I had the pleasure of interning at FAIR (Meta). I would like to thank my mentor, Fabio Petroni, who, along with Luke Zettlemoyer and Sebastian Riedel, deeply influenced my research ideas and taste, as evidenced by my continued collaboration with Fabio on one of my first PhD papers. My time at Microsoft Research was equally enriching, where conversations with my mentor, Eric Horvitz, collaborators Adam Fourney and Gagan

Bansal, and other researchers like Andrey Kolobov helped me develop invaluable insights and research perspectives.
This work was made possible through the support of several institutions. I gratefully acknowledge EPFL for facilitating the doctoral program and awarding me the EDIC PhD fellowship. I would also like to thank the Swiss Data Science Center for their PhD fellowship support, and Microsoft and Google for their generous research grants.
Finally, I would like to thank my parents, Miroslav and Marina, and my older sisters, Maja and Mimi, for their unconditional support and for providing me with a solid foundation to build upon. Beyond academic life, my PhD journey led me to my wife, Eleni, whose constant love makes me happier with each passing day. For this, I am grateful, and this alone makes me certain that the journey was more than worth it.

Abstract

Developing agents that can reliably act on our behalf is central to artificial intelligence (AI). These agents must seamlessly interact with tools, like search engines and databases, and collaborate. In this thesis, we study the abstractions, methods, and infrastructure needed to enable and support the development of AI agents in the era of large language models (LLMs). The contributions of the thesis are divided into four parts.

Part 1 examines goal-oriented collaboration between two components, at least one of which is LLM-based. For an LLM-based component to interact successfully with others, it must adhere to specified interfaces, especially when interacting with traditional software-based components exposed through an API, and steer the collaboration toward high-utility outcomes. We show that LLM decoding algorithms serve as an efficient strategy to accomplish both objectives without modifying the underlying model.

Part 2 focuses on scenarios where the underlying model's capabilities are insufficient for effective collaboration, and the training signal necessary for improving the model is not readily available. To address this challenge, we introduce the principle of exploiting asymmetry for synthetic data generation and demonstrate how it can be applied to generate useful data even for tasks that LLMs cannot solve directly. We highlight the generality of this approach by drawing connections to seminal work on self-improvement for LLMs.

Part 3 addresses the collaboration among multiple AI systems, tools, and humans. We propose an abstraction that, in concert with the accompanying library, provides a theoretical and practical infrastructure with a modular and concurrency-friendly design, which enables the modeling, implementation, and systematic study of arbitrarily complex structured interactions. To demonstrate the potential of the framework and the accompanying library, we use them to systematically investigate the benefits of complex interactions for solving competitive coding problems.

Part 4 proposes a novel perspective called semantic decoding that allows us to systematically study the design space of structured interactions. We conclude this part by discussing the research opportunities and questions emerging from the semantic decoding perspective, enabled by the foundation laid in Parts 1, 2, and 3.

Keywords: AI agents, large language models, synthetic data generation, decoding algorithms, transformers, natural language processing, artificial intelligence.

Résumé

Développer des agents capables d'agir de manière fiable en notre nom est au cœur de l'intelligence artificielle (IA). Ces agents doivent interagir sans difficulté avec des outils, comme les moteurs de recherche et les bases de données, et collaborer. Dans cette thèse, nous étudions les abstractions, méthodes et infrastructures nécessaires pour faciliter et soutenir le développement d'agents IA exploitant les grands modèles de langage (LLMs). Les contributions de cette thèse sont divisées en quatre parties.

La première partie porte sur la collaboration entre deux composants, dont au moins un emploie LLM, pour atteindre un objectif commun. Pour que ce composant basé sur un LLM puisse interagir efficacement avec d'autres, il doit respecter des interfaces définies, notamment lorsqu'il s'agit de travailler avec des composants logiciels traditionnels accessibles via une API, et orienter la collaboration vers des résultats pertinents. Nous montrons que les algorithmes de décodage des LLMs constituent une stratégie efficace pour atteindre ces deux objectifs sans modifier le modèle sous-jacent.

La deuxième partie aborde des situations où les capacités du modèle sous-jacent sont insuffisantes pour une collaboration efficace, et où les données nécessaires à l'entraînement du modèle ne sont pas facilement disponibles. Pour surmonter cet obstacle, nous introduisons un principe d'exploitation des asymétries pour générer des données synthétiques, montrant comment il peut être utilisé pour produire des données utiles, même pour des tâches que les LLMs ne peuvent pas résoudre directement. Nous soulignons la flexibilité de cette approche en la rattachant à des travaux pionniers sur l'auto-amélioration des LLMs.

La troisième partie se concentre sur la collaboration entre plusieurs systèmes d'IA, outils et humains. Nous proposons une abstraction qui, avec la librairie de code associée, fournit une infrastructure théorique et pratique, modulaire et adaptée à la concurrence, permettant de modéliser, implémenter et étudier systématiquement des interactions structurées de complexité variable. Pour illustrer les possibilités offertes par ce cadre et cette librairie, nous les utilisons pour étudier systématiquement les bénéfices des interactions complexes dans la résolution de problèmes de codage compétitif.

Finalement, la quatrième partie introduit une nouvelle approche, le décodage sémantique, qui permet d'explorer de manière systématique les interactions structurées possibles. Cette partie se termine par une discussion sur les opportunités de recherche et les questions soulevées

par cette perspective de décodage sémantique, rendue possible par les bases posées dans les parties précédentes.

Mots clés : agents IA, grands modèles de langage, génération de données synthétiques, transformeur, traitement automatique des langues, intelligence artificielle.

Contents

List of Figures

List of Tables

Part I Introduction and Background

1 Introduction

1.1 Motivation

The scaling of large language models (LLMs) in terms of data, parameters, and computational resources has fundamentally transformed the artificial intelligence landscape. The remarkable success of these models largely lies in their emergent ability to leverage and adapt to information within their context and map it to appropriate outputs with effectiveness and generalization that other machine-learning approaches struggle to match [24, 268, 127]. By carefully engineering the context, LLMs can be conditioned to effectively perform complex reasoning [268, 184] and leverage external tools [169], substantially expanding and enhancing their capabilities. This has enabled LLMs to achieve state-of-the-art performance across domains ranging from sophisticated code generation [37] to protein folding prediction [121].

While LLMs have demonstrated remarkable capabilities, their fundamental mode of operation is static—they receive input and map it to an output. This limits their utility to an "advisory" role, requiring humans to interpret and manually act on the information in their generated outputs. Agents, in contrast, can directly act on our behalf—autonomously executing actions, monitoring outcomes, and adapting their strategies [214]. This ability not only reduces the human operational overhead but removes this particularly restrictive bottleneck, enabling rapid response times and continuous optimization through direct experience. LLMs present a compelling foundation for digital agents due to their strong generalization capabilities. By treating APIs as tools and strategically prompting the model, LLMs can act in digital environments through API calls and get transformed into agents. Given that much of contemporary knowledge work happens on computers, such digital agents could automate or semi-automate many knowledge workers' daily tasks. Consequently, the development of LLM-based agents emerged as a central area of AI research.

The iterative prompting of an LLM-based agent can enable basic agentic capabilities, but it has proven insufficient for building autonomous agents. Several benchmarks targeting different aspects of agent capabilities have been proposed to systematically evaluate and track progress. These benchmarks range from those assessing complex reasoning and tool use, such

as GAIA (General AI Agent benchmark), to domain-specific evaluations like SWE-bench and MLE-bench that focus on software engineering tasks [170, 116, 31]. Notably, the current state-of-the-art performance on these benchmarks is achieved by AI systems defined by carefully orchestrated interaction between multiple components of differently prompted LLMs and tools. However, these are hand-crafted and offer only a glimpse into the vast potential of structured interactions.

To realize this potential, we must develop principled ways of studying the structured interactions underpinning such compound AI systems. However, the design space is incredibly vast, spanning multiple levels: (1) the LLMs; (2) the integration of the LLMs into two-component interactions; (3) the structured interactions characterizing the system. This thesis focuses on enabling and supporting the principled development of compound AI systems and addresses fundamental challenges at each level.

1.2 Thesis Overview and Contributions

The thesis is organized into eight chapters divided into five parts. Part I provides an introduction and general background. Parts II, III, and IV form the core of the thesis, addressing fundamental challenges in the development of compound AI at different levels of the design space. In Part V, we propose a perspective that allows us to systematically study the design space of structured interactions based on the optimization they perform; we discuss the research opportunities emerging from our work and conclude the thesis.
Each part begins with a preface that contextualizes the work presented in the comprising chapters and connects it to the broader theme of the thesis. Each chapter concludes with a section providing a summary and a discussion of the work's implications, impact, and limitations.

1.2.1 Part II: Integrating LLMs In Collaborations

In this part, we study the question of how to effectively integrate an LLM as part of a collaboration without modifying the model. Specifically, two requirements must be met for a collaboration between two components to be successful. First, each component needs to adhere to the other's interface. Second, their interactions should provide a useful signal toward achieving a high-utility outcome for the collaboration.

Constrained Decoding for Explicit Alignment. Chapter 3 shows how constrained decoding is an effective strategy for ensuring that the output of an LLM follows an explicitly specified schema. We focus on closed information extraction (cIE), the problem of extracting structured semantic information from unstructured texts. This task exemplifies the challenge of mapping an input to an output from a very large structured output space and has been well-studied in the field of IE. We introduce GenIE (generative information extraction), the first end-to-end autoregressive formulation of cIE. GenIE leverages the autoregressive formulation to capture the fine-grained interactions expressed in the text and employs a constrained generation

strategy to ensure the outputs are always consistent with the predefined knowledge base schema. Our experiments show that GenIE is state-of-the-art on cIE, generalizes from fewer training data points than baselines, and scales to a previously unmanageable number of entities and relations. This chapter is based on Josifoski et al. [119].

Decoding as Misalignment Mitigation. Chapter 4 studies decoding algorithms as a tool for aligning the models' output likelihood with task-specific utility more broadly. A critical component of a successful LLM generation pipeline is the *decoding algorithm*. However, the general principles that should guide the choice of a decoding algorithm remain unclear. Previous works only compare decoding algorithms in narrow scenarios, and their findings do not generalize across tasks. We argue that the misalignment between the model's *likelihood* and the task-specific notion of *utility* is the key factor to understanding the effectiveness of decoding algorithms. To structure the discussion, we introduce a taxonomy of misalignment mitigation strategies (MMSs), providing a unifying view of decoding as a tool for alignment. The MMS taxonomy groups decoding algorithms based on their implicit assumptions about likelihood–utility misalignment, yielding general statements about their applicability across tasks. Specifically, by analyzing the correlation between the likelihood and the utility of predictions across a diverse set of tasks, we provide empirical evidence supporting the proposed taxonomy and a set of principles to structure reasoning when choosing a decoding algorithm. Crucially, our analysis is the first to relate likelihood-based decoding algorithms with algorithms that rely on external information, such as value-guided methods and prompting, and covers a much more diverse set of tasks than prior work. This chapter is based on Josifoski et al. [120].

1.2.2 Part III: Training LLMs to Be Collaborative

Part II shows how decoding algorithms can serve as an efficient strategy for integrating LLMs in collaborations without modifying the underlying model. For some tasks, this approach may not be sufficient, necessitating fine-tuning. A challenge arises when the training signal required for such improvement is not readily available. In this part, we investigate how synthetic data generation can be used to address this challenge.

Asymmetry for Synthetic Data Generation. Chapter 5 shows that useful data can be synthetically generated even for tasks that cannot be solved directly by LLMs: for problems with structured outputs, it is possible to prompt an LLM to perform the task in the reverse direction, by generating plausible input text for a target output structure. Leveraging this asymmetry in task difficulty makes it possible to produce large-scale, high-quality data for complex tasks. We demonstrate the effectiveness of this approach on cIE, where collecting ground-truth data is challenging, and no satisfactory dataset exists to date. We synthetically generate a dataset of 1.8M data points, establish its superior quality compared to existing datasets in a human evaluation, and use it to fine-tune small models that outperform the prior state of the art (with equal model size) by a margin of 57 absolute points in micro-F1 and 79 points in macro-F1.

While we highlight the efficacy of leveraging asymmetries by developing a specific pipeline for cIE, the approach is general, and any problem with an inverse formulation that can be addressed more effectively by the LLM can benefit from it. This chapter is based on Josifoski et al. [117]

1.2.3 Part IV: Developing Collaborations

The preceding chapters focused on methods ensuring that LLMs can be effectively integrated into two-component collaborations. This ability opens up unprecedented opportunities for structured reasoning as well as collaboration among multiple AI systems, tools, and humans. In this part, we introduce a conceptual framework and an accompanying library that enable and support the development of such collaborations of arbitrary complexity.

Flows: Building Blocks of Reasoning and Collaborating AI. Chapter 6 introduces the conceptual framework *Flows*. Flows are self-contained building blocks of computation, with an isolated state, communicating through a standardized message-based interface. This modular design simplifies the process of creating Flows by allowing them to be recursively composed into arbitrarily nested interactions and is inherently concurrency-friendly. Crucially, any interaction can be implemented using this framework, including prior work on AI–AI and human–AI interactions, prompt engineering schemes, and tool augmentation. We demonstrate the potential of *Flows* on competitive coding, a challenging task on which even GPT-4 struggles. Our results suggest that structured reasoning and collaboration substantially improve generalization, with AI-only Flows adding +21 and human–AI Flows adding +54 absolute points in terms of solve rate. To support rapid and rigorous research, we introduce the `aiFlows` library embodying the abstraction. This chapter is based on Josifoski et al. [118]

1.2.4 Part V: Outlook and Conclusion

Flows as Semantic Decoding Algorithms. Chapter 7 proposes the semantic decoding perspective that offers a powerful abstraction for search and optimization directly in the space of meaningful concepts. Specifically, we conceptualize LLMs as semantic processors that manipulate meaningful pieces of information that we call semantic tokens. LLMs are among a large pool of other semantic processors, including humans and tools, such as search engines or code executors. Collectively, semantic processors engage in dynamic exchanges of semantic tokens to progressively construct high-utility outputs. We refer to these orchestrated interactions among semantic processors, optimizing and searching in semantic space, as *semantic decoding algorithms*. We formalize the transition from syntactic to semantic tokens and the analogy between syntactic and semantic decoding. Subsequently, we explore the possibilities of optimizing within the space of semantic tokens via semantic decoding algorithms. By focusing on the semantic level and disregarding syntactic details, we gain a fresh perspective on the engineering of AI systems, enabling us to imagine systems with much greater complexity

and capabilities. We conclude the chapter with a list of research opportunities and questions emerging from the work of this thesis. This chapter is based on our position paper [195].

2 Background

2.1 AI Agents

An agent can be formally defined as an autonomous system that perceives its environment, processes information, and takes actions in pursuit of goals or objectives, with the ability to adapt its strategies based on feedback. Developing autonomous agents that can reliably act on our behalf is one of the core problems of AI [214].

Traditional approaches to AI agent development have primarily relied on two paradigms: symbolic systems and reinforcement learning. Symbolic AI, often referred to as Good Old-Fashioned AI, employs explicitly encoded knowledge and logical reasoning to create agents that operate through predefined rules and cognitive architectures [181]. Systems like Soar and ACT-R exemplify this approach by integrating multiple cognitive modules to emulate human-like reasoning [133, 5]. However, these rule-based agents often struggle to scale beyond controlled environments and adapt to complex, real-world situations.

To address these limitations, researchers turned to Reinforcement Learning (RL), which enables agents to learn optimal behaviors through environmental interaction [214]. While RL has achieved remarkable results in specific domains such as game playing and robotic control [226, 112], these successes come at a cost. RL agents typically require extensive training data and computational resources, and their learned behaviors rarely generalize beyond their training domain [134].

The limitations of both symbolic and RL approaches highlight a fundamental challenge in agent development: creating systems that can generalize across diverse tasks and adapt to novel situations without extensive retraining.

2.2 Static LLMs to LLM-Based Agents

A language model defines a probability distribution p over $\mathbf{y} \in \mathscr{Y}$, where $\mathscr{Y}$ is the set of all sequences that can be constructed using a vocabulary $\mathscr{V}$. To efficiently represent a probability

distribution over the large combinatorial space of all possible strings, language models use an auto-regressive decomposition—a strategy that predicts each token given all previous tokens in the sequence: $p_\theta(y_t|\mathbf{y}_{<t})$, where $\mathbf{y}_{<t} := \langle y_0, \ldots, y_{t-1} \rangle$. Most modern language models are parameterized by a Transformer architecture with trainable weights θ [257]. Then, the probability distribution of an output sequence $\mathbf{y} := \langle y_0, \ldots, y_n \rangle$, potentially conditioned on an input $\mathbf{x} := \langle x_0, \ldots, x_m \rangle$, is given by

$$p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=0}^{|\mathbf{y}|} p_\theta(y_t|\mathbf{y}_{<t}, \mathbf{x}). \tag{2.1}$$

The model is trained to maximize the target sequence's conditional log-likelihood with teacher forcing, using the cross-entropy loss $\mathscr{L}(\theta) = -\sum_{(\mathbf{x},\mathbf{y})\in\mathscr{D}} \log p_\theta(y \mid x)$, where $\mathscr{D}$ is the training corpus [244, 245].

The scaling of large language models has revealed a remarkable emergent property: as these models grow in size and training data, they develop an increasingly sophisticated ability to generalize from information provided in their context [24, 127, 268]. This generalization manifests itself as a fundamental capacity to comprehend and adapt to (potentially novel) patterns and instructions presented within their context window. By leveraging this emergent capability through careful prompt engineering—the strategic construction of input contexts—LLMs have achieved state-of-the-art performance across diverse domains [37, 121]. This breakthrough has effectively transformed LLMs from specialized language processors into general-purpose computational engines that can be dynamically repurposed through context engineering.

One particularly powerful application of this ability to leverage and adapt to information in the context is tool augmentation, where LLMs are prompted to interact with tools external to the model through well-defined APIs [169, 220]. In this approach, models generate language commands that are parsed and executed as API calls, thereby extending their capabilities. By exposing the possible interactions with an environment through APIs, these generated commands become meaningful actions, transforming static language models into interactive agents. Among the first to demonstrate this transformation was WebGPT [178], which augmented a language model with the ability to interact with a simple web browser, enabling it to gather and synthesize information from the web to answer questions. Overall, the combination of strong generalization capabilities and the ability to act through language makes LLMs a compelling foundation for building autonomous agents. Unlike traditional approaches that require explicit programming of action spaces and policies, LLM-based agents can flexibly interpret and generate appropriate actions across diverse domains based on natural language descriptions of tools and objectives.

2.3 Modern AI Agents are Systems of Structured Collaborations

Tool augmentation gives LLMs the ability to act, transforming them into LLM-based agents. Even in this most rudimentary form, LLM-based agents are more than a statistical model—they are AI systems defined by the interactions between a model and external tools. Crucially, the current state-of-the-art performance on every benchmark targeting agent capabilities is achieved by an AI system defined by intricate, carefully orchestrated interactions between multiple components of differently prompted LLMs and tools [170, 116, 31]. In this thesis, we refer to such systems as *collaborative AI systems*.

The paradigm of collaborative AI systems will persist even as individual models become more capable for three fundamental reasons. First, certain capabilities inherently require external augmentation, regardless of the models' capabilities. For instance, applications requiring real-time or private information unavailable during model training must necessarily rely on external systems to provide this information at inference time. Second, it is highly likely that generalist models will continue to benefit from external augmentation in the near future. For a single generalist model to operate without augmentation, it would need to simultaneously outperform every specialized model—whether in protein folding, autonomous driving, or other domains—while also replicating the functionality of external components like processors. A more probable scenario is that advanced generalist models will excel at identifying and leveraging specialized tools to enhance their capabilities in pursuit of specific objectives. Third, a general-purpose LLM is, by definition, a foundational tool used across different domains. When using this general tool for specific applications, developers will often—if not always—be able to improve performance by baking domain knowledge into the AI system's architecture. For instance, current coding agents greatly benefit from the ability to iterate over the code using a compiler and a debugger (see Chapter 6).

Collaborative AI systems offer clear benefits, yet realizing their full potential requires overcoming fundamental challenges. First, we need frameworks that enable rapid design, implementation, and systematic study of complex interaction patterns without sacrificing expressivity—a critical requirement given the vast design space of possible collaborations. Second, executing these interactions reliably and at scale is challenging. Interactions among flexible, probabilistic LLMs, traditional software with rigid interfaces, and specialized components create numerous opportunities for failure. Scaling these collaborations demands robust infrastructure and optimized execution strategies. Finally, optimizing collaborative AI systems requires addressing both the structure of interactions and the individual components. Currently, no learning methods can co-optimize both aspects, and this challenge is exacerbated by the scarcity of training data capturing the intricacies of component interactions.

Integrating LLMs In Collaborations Part II

Preface

Part II examines the fundamental unit of collaboration: the interaction between two components. Concretely, we focus on collaborations initialized with a specific goal, where at least one of the components is LLM-based. A canonical example is a human user interacting with an LLM (Fig. 2.1a) — whether through one or multiple interactions, the goal of the collaboration is for the user to reach a high-utility outcome satisfying his goal. As illustrated in Figure 2.1, this setting extends beyond human-AI collaboration to include (single-turn and multi-turn) tool use (Fig. 2.1b and 2.1c) and AI-AI collaboration (Fig 2.1d).

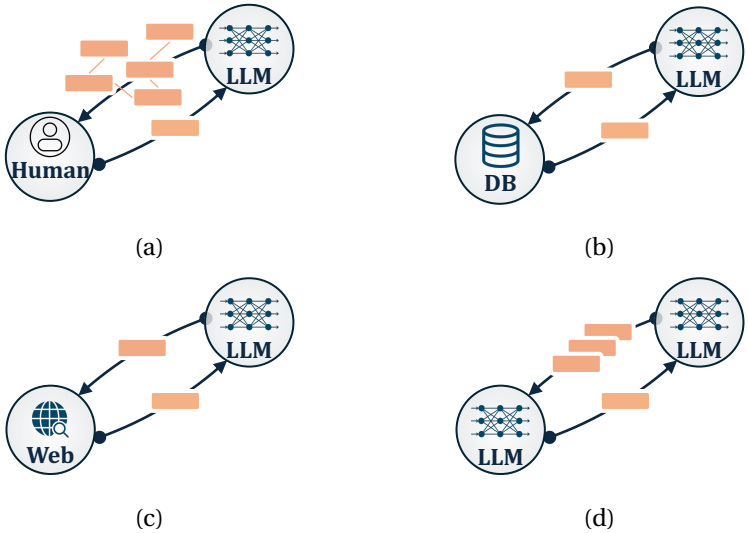


Figure 2.1: **Examples of two-component collaborations.**

Two essential requirements must be met for a collaboration between two components to be successful. First, each component needs to adhere to the other's interface—in other words, they need to "speak the same language." Second, their interactions should provide a useful signal toward achieving a high-utility outcome for the collaboration.

In this part, we study the question of *how to effectively integrate LLMs into collaborations without modifying the model.* We conceptualize both requirements as alignment of the LLM's output, corresponding to each interaction, to the collaboration-specific utility function. Indeed, failing to adhere to interfaces breaks the collaboration, resulting in zero utility, especially when interacting with traditional software-based components exposed through an API or specialized models. Chapter 3 shows how constrained decoding is an effective strategy for ensuring that LLM outputs follow an explicitly specified schema. Chapter 4 generalizes this notion and studies decoding algorithms as a tool for alignment more broadly.

3 Constrained Decoding for Explicit Alignment

3.1 Introduction

The ability to extract structured semantic information from unstructured texts is crucial for many AI tasks such as knowledge discovery [113, 254], knowledge maintenance [248], symbolic representation, and reasoning [114]. The interface between free text and structured knowledge was formalized by *knowledge base population* [KBP; 113], which proposes to represent the information contained in text using *(subject, relation, object)* fact triplets. In this chapter, we focus on closed information extraction (cIE), the problem of extracting exhaustive sets of fact triplets expressible under the relation and entity constraints defined by a Knowledge Base (KB) schema. We choose this task as it exemplifies the challenge of mapping an input to an output from a very large structured output space. It is worth noting that when this work was conducted, available language models completely failed the task without dedicated training or fine-tuning, so the models used in this section were specifically trained for it. In follow-up work, we extend the ideas proposed in this section to scenarios using LLMs without fine-tuning. We revisit this in Sec. 3.6.

Traditionally, cIE was approached with pipelines that sequentially combine named entity recognition [251], entity linking [173], and relation extraction [172]. Entity linking and relation extraction serve as grounding steps, matching entities and relations to numerical identifiers in a KB, e.g., QIDs and PIDs for Wikidata [259]. Recently, Trisedya et al. [254] pointed out that such pipeline architectures suffer from the accumulation of errors and proposed an end-to-end alternative. Nevertheless, existing methods are still only practical for small schemas with unrealistically small numbers of relations and entities.

Alternatively, some works have focused on a simpler syntactic task: open information extraction (oIE), which produces free-form triplets from texts. In this setup, the entities and relations are not grounded in a KB and, usually, do not represent facts [82]. As oIE triplets contain only surface relations, they have ambiguous semantics, making them hard to use in downstream tasks [23] if not first aligned with a KB [82]. Since, in practice, oIE often consists of structured substring selection, it has recently been framed as an end-to-end sequence-to-sequence

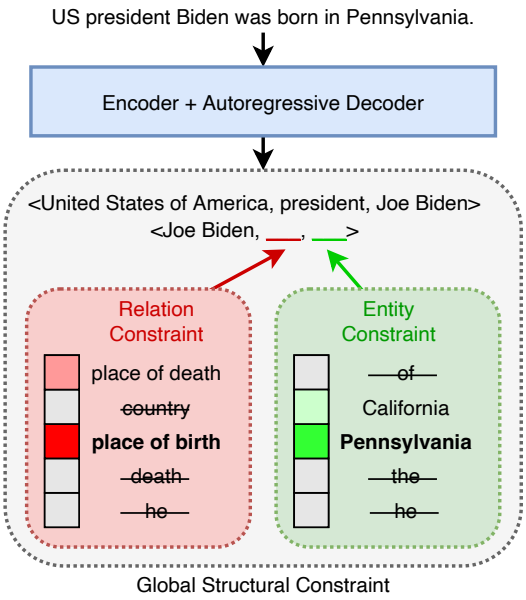


Figure 3.1: **Overview of GenIE.** We use a transformer encoder-decoder model that takes unstructured text as input and autoregressively generates a structured semantic representation of the information expressed in it in the form of (subject, relation, object) triplets. GenIE employs constrained beam search with (i) a high-level constraint that asserts that the output corresponds to a set of triplets and (ii) lower-level constraints that use prefix tries to force the model to only generate valid entity or relation identifiers (from a predefined schema).

problem with great success [111, 60]. Indeed, such autoregressive formulations can exploit the language knowledge already encoded in pre-trained transformers [58]. For example, some tokens can be more easily recognized as possible entities or relations thanks to the pre-training information.

Inspired by recent successes in oIE, we propose the first autoregressive end-to-end formulation of cIE that scales to many entities and relations, making cIE practical for more realistic KB schemas (*i.e.* schemas with millions of entities).[1] We employ a sequence-to-sequence BART model [140], and exploit a novel bi-level constrained generation strategy operating on the space of possible triplets (from a fixed schema induced by Wikidata) to ensure that only valid triplets are generated. Our resulting model, *GenIE*, performs *Gen*erative *I*nformation *E*xtraction and combines the advantages of a known schema with an autoregressive formulation. The high-level overview of GenIE is provided in Fig. 3.1. The constrained generation encodes the known schema and enables the autoregressive decoder to generate textual tokens but only from the set of allowed entities or relations.

Contributions. (i) We present the first end-to-end autoregressive formulation of closed information extraction. (ii) We describe a constrained decoding strategy that exploits the

[1] Note that prior methods, due to the atomic classification, have high memory requirements, and suffer from performance deterioration as the number of entities or/and relations grows.

Wikidata schema to generate only valid fact triplets, demonstrating how constrained beam search can be applied on large, structured, and compositional spaces. (iii) We propose a model that achieves state-of-the-art performance on the cIE task and scales to previously unmanageable numbers of entities (6M) and relations (more than 800). (iv) We point out and address weaknesses in the evaluation methodologies of recent previous works stemming from their small scale and the large imbalances in the available data per relation. We demonstrate the importance of reporting performance as a function of the number of relation occurrences in the data. (v) We release pre-processed data, pre-trained models, and code within a general template designed to facilitate future research at https://github.com/epfl-dlab/GenIE.

3.2 Background and Related Work

3.2.1 Closed Information Extraction

In this work, we address the task of closed information extraction (cIE), which aims to extract the exhaustive set of facts from natural language, expressible under the relation and entity constraints defined by a knowledge base (KB).

Most of the existing methods address the problem with a pipeline solution. One line of work starts by first extracting the entity mentions and the relations between them from raw text. This is followed by a disambiguation step in which the entity and relation predicates are mapped to their corresponding items in the KB. The sub-task of extracting the free-form triplets was originally proposed by Banko et al. [15], and it is commonly referred to as open information extraction (oIE) or text-to-graph in the literature [92, 28, 111, 228]. Another line of work employs a pipeline of models for (i) named entity recognition (NER) — detecting the entity mentions; (ii) entity linking (EL) — mapping the mentions to specific entities from the KB; (iii) relation classification (RC) — detecting the relations that are expressed between the entities [73, 8, 30]. Due to their architecture, pipeline methods are plagued by error propagation, which significantly affects their performance [168, 254].

End-to-end systems that jointly perform the extraction and the disambiguation of entities and relations have been proposed to address the error propagation [254, 242, 151]. To mitigate the propagation of errors, these systems are endowed with the ability to leverage entity information in the relation extraction and vice-versa, which has resulted in significant performance gains. Conceptually, for producing the output triplets, existing methods all rely on atomic, multi-class classification-based ranking of relations and entities. Classification methods particularly suffer from imbalances in the data. On the contrary, our model, GenIE, is autoregressive and copes better with imbalances.

While cIE requires the constituent elements of the output triplets to be entities and relations associated with the KB, the output triplets in oIE are free-text. This makes the cIE task fundamentally harder than oIE and renders the majority, if not all, oIE methods inapplicable to the

cIE setting. We report an additional discussion on relevant, but not fundamental, related work on oIE in Appendix A.1.

3.2.2 Autoregressive Entity Linking

The tasks of entity linking (EL) and entity disambiguation (ED) have been extensively studied in the past [109, 276, 136, 128, 10]. Most existing approaches associate entities with unique atomic labels and cast the retrieval problem as multi-class classification across them. The match between the context and the label can then be represented as the dot product between the dense vector encodings of the input and the entity's meta information [276]. This general approach has led to large performance gains.

Recently, De Cao, Aziz, and Titov [51] and De Cao et al. [52, 53] have suggested that the classification-based paradigm for retrieval comes with several shortcomings such as (i) the failure to capture fine-grained interactions between the context and the entities; (ii) the necessity of tuning an appropriately hard set of negative samples during training. Building on these observations, they propose an alternative solution that casts the entity retrieval problem as one of autoregressive generation in which the entity names are generated token-by-token in an autoregressive fashion. The (freely) generated output will not always be a valid entity name, and to solve this problem De Cao et al. [52] propose a constrained decoding strategy that enforces this by employing a prefix trie. Their method scales to millions of entities, achieving state-of-the-art performance on monolingual and multilingual entity linking.

Inspired by the intuition that language models are well suited for predicting entities, we propose a novel approach for cIE by framing the problem in an autoregressive generative formulation.

3.3 Method

In this section, we formalize GenIE, an autoregressive end-to-end model for closed information extraction. Let us assume a knowledge base (KB) consisting of a collection of entities $\mathscr{E}$, a collection of relations $\mathscr{R}$, and a set of facts $(s, r, o) \in \mathscr{E} \times \mathscr{R} \times \mathscr{E}$ stored as (subject, relation, object) triplets. Additionally, we assume that each entity $e \in \mathscr{E}$ and relation $r \in \mathscr{R}$ is assigned to a textual label (corresponding to its name). The Wikidata KB [259], with Wikipedia page titles as entity names, and the Wikidata relation labels as relation names, satisfy these assumptions.

3.3.1 Model

We cast the task of information extraction as one of autoregressive generation. More concretely, given some text input x, GenIE strives to generate the linearized sequence representation y of the exhaustive set of facts expressed in x. The conditional probability (parameterized by θ) assigned to the output y is computed in the autoregressive formulation:

$p_\theta(y \mid x) = \prod_{i=1}^{|y|} p_\theta(y_i \mid y_{<i}, x)$. This can be seen as translating the unstructured text to a structured, unambiguous representation in a sequence-to-sequence formulation. GenIE employs the BART [140] transformer architecture. It is trained to maximize the target sequence's conditional log-likelihood with teacher forcing [244, 245], using the cross-entropy loss. We use dropout [238] and label smoothing for regularization [246].

3.3.2 Output Linearization

To represent the output with a sequence of symbols that is compatible with sequence-to-sequence architectures, we introduce the special tokens <sub>, <rel>, <obj> to demarcate the start of the subject entity, the relation type and the object entity for each triplet. The special token <et> is introduced to demarcate the end of the object entity, which is also the end of the triplet. We construct the sequence representation by concatenating the textual representations of its constituent triplets. While the sequence representation has an intrinsic notion of order, the output set of triplets does not. To mitigate the effects of this discrepancy, we enforce a consistent ordering of the target triplets during training. Concretely, whenever the triplets' entities are linked to the entity mentioned in the textual input, we consider first the triplets for which the subject entity appears earlier in the text. Ties are resolved by considering the appearance position of the object entity.

3.3.3 Inference with Constrained Beam Search

The space of triplets corresponds to $\mathscr{T} = \mathscr{E} \times \mathscr{R} \times \mathscr{E}$, and the target space, which consists of triplet sets of arbitrary cardinality, is equivalent to $\mathscr{S} = \bigcup_{i=0}^{\infty} [\mathscr{E} \times \mathscr{R} \times \mathscr{E}]^i$. At inference time, GenIE tackles the task of retrieving the linearized representation $y_S \in \mathscr{S}$ of a set of facts $S = \{t_1, \ldots, t_n\}$ constituted by triplets $t_i \in \mathscr{T}$ expressed in the input text x. Ideally, we would consider every element $y \in \mathscr{S}$ in the target space, assign it a score $p_\theta(y \mid x)$, and retrieve the most probable y. Unfortunately, this is prohibitively expensive since we are dealing with a compositional target space whose size is gigantic (*e.g.*, if we consider a Wikidata entity catalog of $|\mathscr{E}| \approx 6$M elements and a relation catalog of $|\mathscr{R}| \approx 1000$ relations, that can express a total of $|\mathscr{T}| \approx 10^{15}$ triplets; even if we limit ourselves to sentences that express only two facts, this provides us with $\approx 10^{30}$ different output options).

On the other hand, the output needs to follow a particular structure, and contain only valid entity and relation identifiers. This does not necessarily hold for an arbitrary generation from a sequence-to-sequence model.

GenIE employs constrained beam search [BS; 245, 52] to resolve both of these problems. Instead of explicitly scoring all of the elements in the target space $\mathscr{S}$, the idea is to search for the top-k eligible options, using BS with k beams and a prefix trie. BS considers one step ahead – the next token to be generated – conditioned on the previous ones. The prefix trie restricts the BS to candidate tokens that could lead to valid identifiers. However, for the cIE

setting we are interested in, the target space is prohibitively large to pre-compute the necessary trie. Therefore, we enforce a bi-level constraint on the output that allows for compositional, dynamic generation of the valid prefixes. More specifically, GenIE employs: (i) a high-level structural constraint which asserts that the output follows the linearization schema defined in Sec. 3.3.2; (ii) lower level validity constraints which use an entity trie and a relation trie to force the model to only generate valid entity or relation identifiers, respectively — depending on the specific element of the structure that is being generated. This outlines a general approach for applying BS to search through large compositional structured spaces.

3.4 Experimental Setup

3.4.1 Knowledge Base: Wikidata

We use Wikidata[2] [259] as the target KB to link to, filtering out all entities that do not have an English Wikipedia page associated with them. The filtering guarantees that all entity names are unique. Our final entity set $\mathscr{E}$ contains 5,891,959 items. We define our relation set $\mathscr{R}$ as the union of all the relations considered in the datasets described below, resulting in 857 relations. For different datasets, we consider only the subset of annotated relations to better compare with baselines. Although large, the number of entity (and relation) names is not a memory bottleneck as the generated prefix trie occupies ≈200MB of storage (*e.g.*, the entity linking system proposed by Wu et al. [276] needs >20 times more storage).

3.4.2 Datasets and Evaluation Metrics

In this work, we further annotate and adapt REBEL [111] and Wiki-NRE for training, validation and testing. Additionally, we use Geo-NRE [254], and FewRel [93] for testing purposes only. Appendix A.2 contains descriptions of these datasets and their statistics. We measure the performance in terms of micro and macro precision, recall and F1. See Appendix A.3 for a detailed and formal description of these metrics. We also report a 1-standard-deviation confidence interval constructed from 50 bootstrap samples of the data.

3.4.3 Baselines

We compare GenIE against Set Generation Networks [SetGenNet; 242] which is, to the best of our knowledge, the strongest model on Wiki-NRE and Geo-NRE. Note that the authors did not release code or the model and there is no other model from the literature trained and evaluated on REBEL for cIE. SetGenNet [242] is an end-to-end state-of-the-art model for triplet extraction. It consists of a transformer encoder [256] that encodes the input followed by a non-autoregressive transformer decoder [90]. The decoder generates embeddings that are used to predict entities and relations. SetGenNet further uses candidate selection [75, 128]

[2] Dumps from 2019/08/01

	Small Evaluation Schema						*Large Evaluation Schema*			
	Wiki-NRE			Geo-NRE			REBEL			FewRel
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Recall
Micro										
SetGenNet (W)	82.75 ± 0.11	77.55 ± 0.27	80.07 ± 0.27	86.89 ± 0.51	85.31 ± 0.47	86.10 ± 0.34	–	–	–	–
SotA Pipeline (W)	67.43 ± 0.28	54.22 ± 0.21	60.11 ± 0.22	64.60 ± 1.46	64.05 ± 1.46	64.32 ± 1.45	–	–	–	–
SotA Pipeline (R)	50.78 ± 0.20	62.17 ± 0.24	55.90 ± 0.20	60.28 ± 1.45	60.78 ± 1.49	60.53 ± 1.45	43.30 ± 0.15	41.73 ± 0.13	42.50 ± 0.13	17.89 ± 0.24
SotA Pipeline (R+W)	65.17 ± 0.27	54.40 ± 0.20	59.30 ± 0.21	66.65 ± 1.47	66.22 ± 1.46	66.43 ± 1.45	–	–	–	–
GenIE (W)	88.18 ± 0.13	88.31 ± 0.16	88.24 ± 0.13	86.46 ± 1.05	87.14 ± 1.03	86.80 ± 1.03	–	–	–	–
GenIE (R)	27.98 ± 0.13	67.16 ± 0.20	39.50 ± 0.14	39.69 ± 1.65	59.01 ± 1.56	47.45 ± 1.62	68.02 ± 0.15	69.87 ± 0.14	68.93 ± 0.12	30.77 ± 0.27
GenIE (R+W)	**91.39 ± 0.15**	**91.58 ± 0.14**	**91.48 ± 0.12**	**91.77 ± 0.98**	**93.20 ± 0.83**	**92.48 ± 0.88**	–	–	–	–
Macro										
SotA Pipeline (W)	11.96 ± 0.72	10.73 ± 0.46	10.56 ± 0.43	24.82 ± 3.61	22.54 ± 3.67	20.39 ± 2.72	–	–	–	–
SotA Pipeline (R)	19.39 ± 1.18	17.41 ± 0.99	15.93 ± 0.93	28.80 ± 3.86	30.24 ± 4.46	25.24 ± 3.21	12.20 ± 0.35	10.44 ± 0.22	9.48 ± 0.21	19.67 ± 0.26
SotA Pipeline (R+W)	24.12 ± 1.46	16.55 ± 1.00	17.76 ± 1.01	38.67 ± 5.72	34.49 ± 5.99	35.14 ± 5.09	–	–	–	–
GenIE (W)	44.22 ± 2.40	36.79 ± 1.62	38.39 ± 1.71	57.13 ± 6.83	52.83 ± 6.84	52.79 ± 6.27	–	–	–	–
GenIE (R)	30.63 ± 1.40	41.97 ± 1.92	29.27 ± 1.26	32.38 ± 5.86	40.39 ± 5.17	30.67 ± 5.23	33.90 ± 0.73	30.48 ± 0.65	30.46 ± 0.62	30.78 ± 0.26
GenIE (R+W)	**52.55 ± 2.12**	**45.95 ± 1.67**	**47.08 ± 1.68**	**75.77 ± 7.80**	**71.60 ± 7.95**	**72.59 ± 7.32**	–	–	–	–

Table 3.1: **Main results.** "R" indicates training on REBEL, and "W" training on Wiki-NRE.

to reduce the output space and a bipartite matching loss that handles different prediction orderings (*i.e.*, it generates a set). Note that there are weaker baselines (e.g., Trisedya et al. [254]) we could have used to compare on REBEL, but we were not able to reproduce their code. We report details on the effort made to use these baselines in Appendix A.4.

We also implement a pipeline baseline, consisting of 4 independent steps, namely: (i) *named entity recognition* (NER), which selects the spans in the input source likely to be entity mentions; (ii) *entity disambiguation* (ED), which links mentions to their corresponding identifiers in the KB; (iii) *relation classification* (RC), which predicts the relation between a given pair of entities, and finally; (iv) *triplet classification* (TC), which predicts whether a given triplet is actually entailed by the context. TC is necessary because the previous step (RC) predicts a relation for every pair of entities. Each step needs to be trained independently with a specific architecture tailored for the task, and we made an optimal choice for each step. For the NER component we used the state-of-the-art tagger FLAIR[3] [2], while for ED we used the GENRE linker[4] [52]. These two models were already trained, and we use them for inference only. For RC and TC, we trained a RoBERTa [150] model with a linear classification layer on top (as these two sub-tasks are typically cast as classification problems). Trisedya et al. [254] also proposed many other pipeline baselines but ours outperforms them (see Table A.3 in Appendix A.7 for comparison).

3.5 Results

3.5.1 Performance Evaluation

Models performing cIE can base their predictions on different schemas. In this section, we distinguish between a *small* and a *large evaluation schema.* The *small evaluation schema* is

[3]https://github.com/flairNLP/flair

[4]https://github.com/facebookresearch/GENRE

	REBEL			FewRel
	Precision	Recall	F1	Recall
Micro				
GenIE	**68.02 ± 0.15**	69.87 ± 0.14	68.93 ± 0.12	30.77 ± 0.27
GenIE - PLM	59.32 ± 0.13	**77.78** ± 0.12	67.31 ± 0.10	**46.95** ± 0.27
GenIE - GENRE	64.14 ± 0.14	76.58 ± 0.11	**69.81 ± 0.10**	**46.62** ± 0.25
GenIE unconstrained	65.30 ± 0.14	67.12 ± 0.12	66.20 ± 0.11	26.15 ± 0.27
Macro				
GenIE	**33.90 ± 0.73**	30.48 ± 0.65	30.46 ± 0.62	30.78 ± 0.26
GenIE - PLM	30.66 ± 0.68	**43.33** ± 0.63	**33.85** ± 0.58	**46.96** ± 0.25
GenIE - GENRE	32.02 ± 0.67	39.14 ± 0.68	**33.40** ± 0.62	**46.63** ± 0.24
GenIE unconstrained	32.25 ± 0.66	27.59 ± 0.53	28.20 ± 0.50	26.14 ± 0.24

Table 3.2: **Ablation study on the weights initialization and the constrained generation strategy.**

consistent with previous approaches where models only have to decide between a small set of relations and entities (the schema induced by Wiki- and Geo-NRE). In the *large evaluation schema*, models use the schema induced by REBEL. Models also use the large evaluation schema of REBEL when tested on FewRel, as a high-quality and challenging recall-based evaluation. We consider 3 training setups for GenIE and the pipeline baseline comprised of SotA components: (i) the training set of Wiki-NRE (W) only, (ii) the training set of REBEL (R) only, and (iii) pre-training on REBEL and fine-tuning on Wiki-NRE (R+W). The implementation details are given in Appendix A.5. We report the macro and micro precision, recall, and F1 in Table 3.1. Unfortunately, as the code for SetGenNet is not available, we cannot compute its macro performance, thus we report the micro scores only.

First, on Wiki-NRE (W), we observe a large and significant F1 improvement of 8 and 28 absolute points obtained by GenIE over SetGenNet and the pipeline baseline, respectively, when trained on the same dataset. Despite the much bigger schema employed by REBEL, pre-training on it and then fine-tuning (R+W), improves the performance on Wiki-NRE and Geo-NRE for 3% and 5%, respectively. This highlights that: (i) GenIE can effectively transfer knowledge across datasets/schemas; (ii) GenIE can quickly adapt to new schemas. Due to its rigid, monolithic relation classifier, the pipeline baseline does not possess these qualities. However, the pre-training does improve its macro scores.

Only the newly developed pipeline baseline and GenIE can scale-up to the larger schema[5], and as expected, this setting is more challenging for both models. However, GenIE still preserves a good F1 score of 68 micro and 34 macro, which is a relative increase of 60% and 320%, respectively, over the baseline. While the pipeline has a steeper drop from micro to macro scores, in general, a significant difference between the two is observed in every setting. This suggests that the models perform better for relations associated with many training

[5] See notes on reproducibility in Appendix A.4.

examples and significantly worse for the rest. These findings call for the fine-grained analysis of performance in Sec. 3.5.2 that partitions the relations according to their occurrence count in the training data. For completeness, we also provide an analysis of performance as a function of the number of relations considered, in Appendix A.6.1.

Finally, on FewRel, recall is the only well-defined metric (see Appendix A.2). In this setting as well, GenIE greatly outperforms the baseline by 13 (micro) and 11 (macro) recall points (micro and macro are close as the dataset is class-balanced).

Ablation study. In Table 3.2 we summarize the results of an ablation study considering the pre-training and the constrained generation. We consider three different starting points: (i) a random initialization; (ii) BART [140] pre-trained language model (PLM); (iii) a pre-trained autoregressive entity retrieval model GENRE [52]. The pre-trained models are better in terms of recall and exhibit a better out-of-domain generalization on FewRel. In contrast, they are slightly worse in terms of precision, which translates to maximum improvement of a single point in F1 on REBEL. Another salient advantage of pre-training is reducing the training steps necessary for achieving good results. Indeed, when starting from GENRE, 3-5k steps are sufficient for competitive performance; starting from a PLM necessitates 5-10k steps; while a random initialization requires 40-50k steps for competitive performance. Additionally, the pre-trained versions converge to a lower validation loss (see Fig. A.2 in Appendix A.7).

To quantify the benefits from the constrained generation, we compare the results attained by the randomly initialized model with and without constraints. In addition to ensuring a structure on the output, the constrained generation strategy results in an increase of 2-3 absolute points in terms of F1.

3.5.2 Analysis of Performance as a Function of the Relation Occurrence Count

The datasets naturally present large imbalances, where few relations occur a lot, but most relations are rare. In the previous section, we already observed a large difference between macro and micro F1 scores of models, indicating that the number of occurrences impacts model performances. Thus, we now measure F1 scores after bucketing relations according to their number of occurrences in the training dataset. In Fig. 3.2, we create buckets $i \in \{0, \dots, 20\}$, where bucket i contains all the relations occurring at least 2^i times and less than 2^{i+1} times in the REBEL dataset. The height of the histogram for bucket i shows how many relations are contained in this bucket. Finally, we report the F1 scores of GenIE and the pipeline of SotA components per bucket. Note that micro F1 from Table 3.1 is equivalent to putting all relations in one single bucket (equal weight to each data point), and macro F1 is equivalent to averaging the F1 with one bucket per relation (equal weight to each relation).

The histogram first confirms that most of the relations occur in only a few triplets from the training data. Models thus need to perform few-shot learning for most of the relations. GenIE is significantly better than the pipeline baseline for all the buckets. Finally, it is important

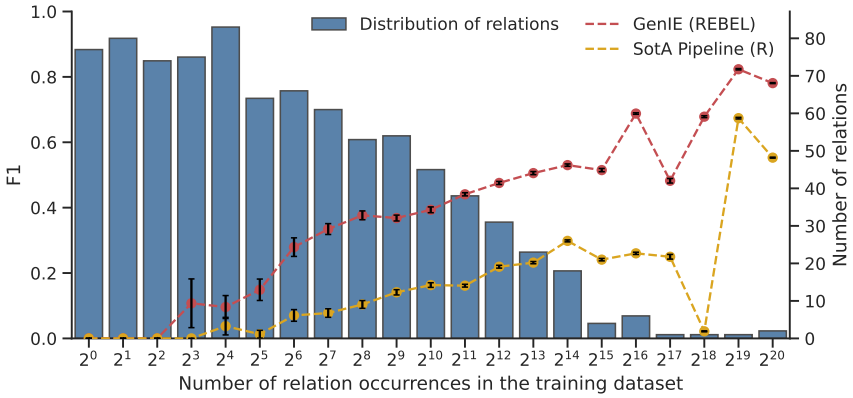


Figure 3.2: **Impact of the number of relation occurrences.** Relations are bucketed based on their number of occurrences; bucket 2^i contains relations occurring between 2^i and 2^{i+1} times. The histogram shows the number of relations per bucket. The line plots depict the F1 scores of GenIE and the baseline per bucket together with confidence intervals computed per bucket with bootstrap resampling.

to highlight that even though the performance of both methods, unsurprisingly, declines for relations that appear less often in the training data, GenIE already performs well for relations with at least $2^6 = 64$ occurrences. On the contrary, the baseline needs $2^{14} = 16{,}384$ samples to reach a comparable level of performance, and scores better than GenIE does for the $2^6 = 64$ bucket only after seeing at least $2^{19} = 524{,}288$ samples. This confirms that GenIE is not only better at macro and micro F1, but it is also capable of performing fewer-shot learning than the baseline. It further shows that, contrary to the baseline, GenIE's good scores do not come solely from its ability to perform well on the few most frequent relations.

3.5.3 Disentangling the Errors

The task of cIE, explicitly or implicitly, encompasses NER, NEL and RC as its subtasks. Failure in any subtask directly translates to failure on the original task. Therefore, to effectively compare different cIE models and accurately characterize their behavior, we need to evaluate their performance on each of the subtasks.

The separation of responsibility between the pipeline components leads to a natural error attribution for the SotA pipeline model. To estimate the NER error, we take the entity mentions predicted by the NER component and compare them with the corresponding mentions of the constituent entities of the output triplets. All triplets that concern an entity whose mention was not retrieved by the NER component are considered erroneous. We differentiate between two settings: (i) exact, which requires that the generated mention exactly matches the target mention; and (ii) partial, for which any overlap between the generated and the target triplet is sufficient. The NEL error is calculated by considering the output of the NEL component

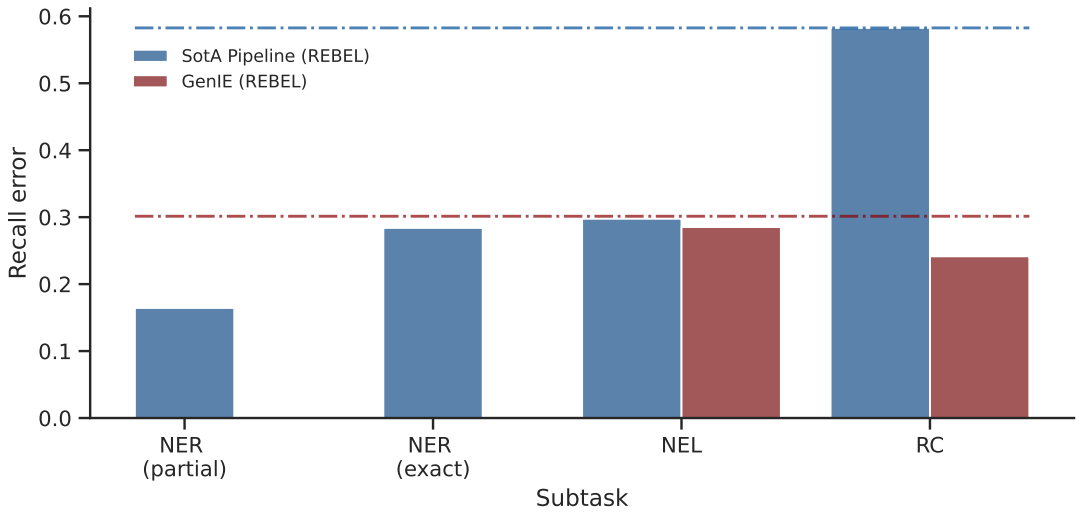


Figure 3.3: **Attribution of error to each of the cIE subtasks.** The dashed lines equal the overall recall error of the system. Lower is better.

and comparing the linked entities to those in the output triplets. Again, all of the triplets that concern an incorrectly linked entity are considered erroneous. Finally, for the pipeline, every correctly predicted relation label translates to a correctly extracted triplet. Therefore, we calculate the RC error using the cIE definition of recall given in Appendix A.3.

End-to-end systems tackle all of the sub-tasks jointly, which makes the error attribution, in this setting, less obvious. To estimate errors corresponding to a particular target triplet, we need a reference triplet (among the predicted ones) for comparison. We start by outlining a bipartite matching procedure. Let each triplet be a node in a graph. We add an edge between each target–prediction pair of triplets. The edges are assigned a weight determined as a function of the pair of triplets they connect. Concretely, an edge e that connects the target triplet $t_T = (s_T, r_T, o_T)$ and the predicted triplet $t_P = (s_P, r_P, o_P)$ will be assigned a weight of: *1*, if the triplets are the same; *2*, if they express the same relation, but either the subject or the object differ; *3*, if they concern the same entities, but the relation differs; *4*, if they share only a single entity; *5*, if they share only a single relation; *6* if they have nothing in common. We construct the matching by selecting edges in a greedy fashion until all of the target triplets have been matched. The procedure ensures that every target triplet is paired with its closest (yet unpaired) match. Finally, we estimate the NEL error as the portion of edges that were assigned a weight $w \in \{2,4,5,6\}$, and the RC error as the portion of edges that where assigned a weight $w \in \{3,4,6\}$.

The results of this analysis are summarized in Fig. 3.3. Immediately, the NER component in the pipeline method introduces an 18% error by completely missing on relevant entity mentions. An additional 12 absolute points hinge on a partial matching. The NEL component matches most of the entity mentions that are retrieved, but at this point, even with a (hypothetical)

perfect RC, the performance of the pipeline will be only on par with GenIE. In practice, the RC component adds 30% to the inherited error, effectively doubling it.

On another note, the absolute error attributed to NEL by the pipeline and GenIE differs in a few absolute points only, while the difference for the (non-inherited) error stemming from RC is less than 10%. Adding these two together leaves us much shorter than the actual gap of 28 absolute points in performance on the cIE task. This gap suggests a strong correlation between the performance on NEL and RC for GenIE, which is fueled by the increased flow of information between the subtasks. The flow of information allows for more fine-grained interactions between the entities, the relations, and the context to be captured, consequently improving the overall performance. Alternatively, whenever the model captures a misleading correlation/interaction, it is amplified and hinders the performance on both subtasks. This result is echoed by the fact that the sum of the errors attributed to NEL and RC is significantly smaller than the error on the cIE task. Based on this observation, we hypothesize that any improvement on NEL will overflow to the RC subtask – and vice-versa – thereby directly translating to performance gains on the overall task.

3.6 Summary and Critical Discussion

Summary. In this chapter, we provide a new view on closed information extraction (cIE) by casting the problem as autoregressive sequence-to-sequence generation. Our method, GenIE, leverages the autoregressive formulation to capture the fine-grained interactions expressed in the text and employs a constrained generation strategy to effectively retrieve the target representation from a large, structured, compositional predefined space of outputs. Experiments show that GenIE achieves state-of-the-art performance on cIE and can scale to a previously unmanageable number of entities and relations. We believe that our autoregressive formulation of cIE, coupled with constrained decoding, is a stepping stone towards a unified approach for addressing the core tasks in information extraction.

Implications for IE. There is a full spectrum of tasks that are closely related to cIE and are central to the field of information extraction. The tasks of entity linking and relation classification, already discussed in Sec. 3.2 and Sec. 3.4.3, are two such examples. Another example is slot filling (SF), the task of extracting information for a specific entity and relation (*e.g.*, entity *Mick Jagger*, relation *member of*) from natural language [243, 193]. All of these problems rely on the same set of logical tasks: identifying items from the KB in the input and understanding how they interact. Therefore, it would be beneficial to assume a single model and collectively solve all of the tasks. This would allow for the information from a dataset collected for one task (*e.g.*, RC) to be leveraged for improving the performance of another (*e.g.*, SF or cIE).
On the other hand, for many useful relations, one of the objects is a literal [168], e.g., date of birth, length, size, number of employees, or others. GenIE can be readily extended to accommodate this by adapting the decoding strategy, allowing the entity to be a substring from the input for specific relations. This is a subtle connection to oIE, which has thus far

been treated as a separate problem. Current state-of-the-art methods on the oIE task address the problem in a similar autoregressive formulation (see Appendix A.1 for a more detailed discussion). In a nutshell, our approach can lead to a unification of the cIE spectrum and bridge the gap between oIE and cIE.

General implications. In this chapter, we demonstrate how constrained decoding and LLMs can effectively map inputs to appropriate outputs within a large compositional structured space. We focus on the well-studied field of IE, ensuring that outputs correspond to (subject, relation, object) triplets, with each element restricted to those from a predefined knowledge base. Building on this success, in follow-up work, we extend these principles to a broader range of tasks using LLMs without fine-tuning [85]. Concretely, we show that by specifying output schemas through grammars and applying constrained decoding, we can effectively adapt LLMs to diverse tasks and output schemas without modifying the underlying model.
The practical value of this approach is evidenced by the several widely used open-source libraries implementing constrained decoding. These include our library GCD [132], as well as LMQL [19, 64], Guidance [171], and SGLang [299, 46], developed by other researchers in the field. Both open-source libraries and commercial APIs, such as OpenAI's GPT [187] and Google's Gemini [88], now offer features to ensure the output is always valid JSON. These implementations vary in their expressiveness and the level of control they provide (*e.g.*., enforcing type constraints to values, supporting recursion, etc.). Currently, research efforts mainly focus on supporting fine-grained JSON schema constraints on the output while minimizing the computation or accuracy overheads associated with applying these constraints [272]. Fundamentally, most of these approaches employ a similar method: running a parser to compute valid continuations at each step. Various optimizations are then applied to this "vanilla" approach. For instance, some methods leverage the grammar's structure to generate multiple tokens simultaneously, similar to speculative decoding, thereby optimizing generation time. Others focus on reducing memory and latency overhead associated with parsing by employing techniques such as lazy automaton generation and memoization.

Challenges and limitations. By restricting the tokens the decoding algorithm can select to those consistent with the predefined constraints, constrained decoding techniques affect the effective output distribution. This approach ensures outputs consistent with imposed grammars or schemas but, potentially, with drastically different likelihoods compared to the unconstrained model. To illustrate this issue, consider an information extraction task on the sentence "Mona Lisa is housed in the Musée du Louvre in Paris." Unconstrained decoding might generate "Mona Lisa, located in, Musée du Louvre" with high probability. However, if the entity catalog contains "Louvre Museum" and "Musée d'Orsay," among others, constrained decoding would force the model to reject "Musée du Louvre," and push the model to a low-probability space towards selecting a less likely option like "Musée d'Orsay", compromising output quality. [189, 86] Indeed, peculiarities associated with the interaction between the syntactic constraints, the model likelihood, and the used decoding algorithms can considerably affect the quality of generated outputs. Another prevalent, seemingly benign issue lies in how

constraints are translated to tokens. For example, when enforcing JSON constraints or valid Python syntax, the original model might be trained on data that predominantly exposes it to tabs (*i.e.*, \t) or 4 spaces. Applying the constraints in a manner inconsistent with the training will adversely affect performance [196].

Current research alleviates these artifacts by developing decoding algorithms and methods that translate the surface form constraints in a way that is informed by the specific tokenization and aligned with the likelihood of the unconstrained model [18, 189]. However, these issues are far from solved. At the core, the fundamental problem is aligning the syntactic (token-level) decoding with the actual semantics associated with the outputs. We revisit this problem in Chapter 7.

While constrained decoding is highly effective in many scenarios, it cannot overcome fundamental limitations in a model's underlying capabilities. These limitations become particularly apparent when integrating models into specialized collaborations. For example, in IE tasks involving proprietary KBs, general-purpose LLMs often struggle because they lack sufficient exposure to specific entity and relation names during pretraining. This challenge extends beyond specialized domains—as shown in Section Sec. 3.5.2, exhaustive information extraction from general text remains beyond current models' capabilities. In such cases, fine-tuning on appropriate training data becomes necessary. Chapter 5 addresses this challenge, focusing specifically on scenarios where the required training signal is not readily available.

4 Decoding as Misalignment Mitigation

4.1 Introduction

Large transformer-based *language models* (LMs) have been pushing the boundaries on tasks ranging from natural language generation [201] to information extraction [119], theorem proving [197], code generation [302], and even protein generation [68]. At inference time, these models rely on a decoding algorithm to generate an output. The goal of decoding algorithms is to select an output of high utility from the exponentially large output space. In contrast to the generic language modeling training objective, which is based on the data likelihood, the notion of utility is task-specific. The potential gap between the two can create a *misalignment* between *model likelihood* and *task utility*; see Fig. 4.1 for an illustration of this concept.

Indeed, across different tasks, researchers noticed that high likelihood is often not associated with desired properties of the output [239, 297, 126]. Naturally, this has led to the development of decoding strategies aimed at mitigating this problem. In the context of natural language generation (NLG), Nucleus Sampling (*i.e.*, top-p) [102] has been proposed to avoid dull or degenerate text. Similarly, in the context of machine translation (MT), solutions ranging from simple ad-hoc tweaks like enforcing a minimal sequence length [239] to leveraging a value model to directly optimize for utility in decoding [137] have been developed. These methods for alignment are task-specific and have been tested only in narrow domains, making it difficult for practitioners to compare them.

Recently, Meister et al. [164] and Wiher, Meister, and Cotterell [271] explored the likelihood–utility misalignment across tasks. However, these studies still largely focus on a group of similar tasks — NLG tasks — and, more crucially, do not include decoding strategies that make use of external sources of information at inference time. Therefore, a general framework to structure our thinking about decoding algorithms is still missing.

Our work makes the first step towards filling this gap. We propose a unified perspective of decoding as a tool for mitigating the likelihood–utility misalignment without modifications

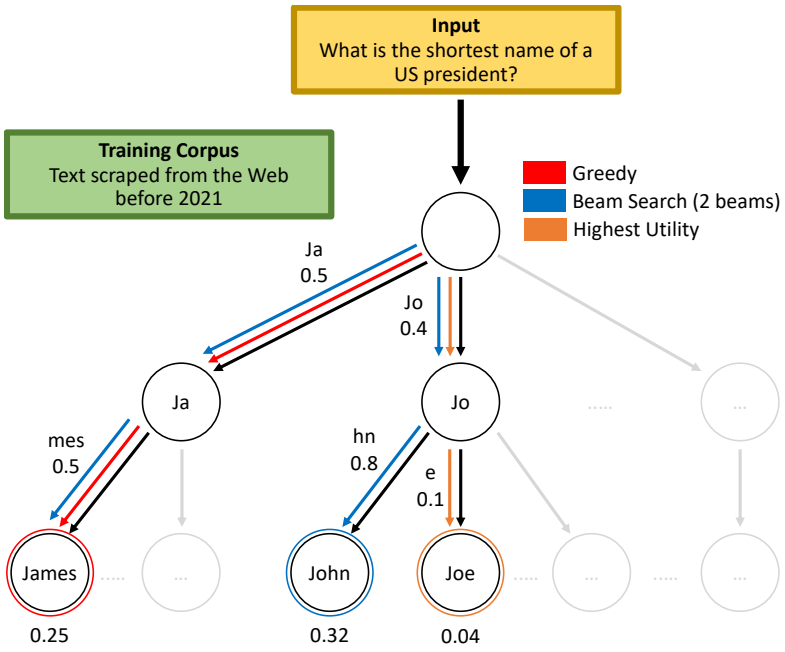


Figure 4.1: **Example of likelihood–utility misalignment.** Imagine a fictional LM trained before Joe Biden became the US president. The input asks for the shortest name of a US president. After Joe Biden's inauguration, this is 'Joe', but before, it was 'John'. Greedy search returns 'James' since its first token 'Ja' has the highest likelihood. Beam search manages to find the highest likelihood sequence 'John'. Both fail to find the correct answer 'Joe' with the highest utility since 'Joe' has a very low likelihood.

to the model. Looking at decoding through this functional lens, in Sec. 4.3, we provide a taxonomy of misalignment mitigation strategies (MMSs). The taxonomy groups decoding algorithms based on the implicit assumptions about likelihood–utility misalignment that need to hold for them to be effective.

Equipped with this taxonomy, we conduct a comprehensive empirical analysis in which we choose a representative set of decoding algorithms and a representative set of tasks to cover the relevant types of misalignment. We identify three main sources of misalignment: *training imperfections* (finite dataset, differentiable surrogate loss), *distribution shift*, and changes in the model's intended usage, which we call *utility drift*. Then, we measure the likelihood–utility misalignment across tasks (RQ1) by estimating: the correlation between likelihood and utility for the generated outputs after decoding (RQ1-a) and the correlation between likelihood and utility among candidate outputs explored by the decoding algorithm (RQ1-b). We proceed by investigating the benefits of decoding algorithms that leverage external information at inference (RQ2). Finally, we experiment with large generalist LMs (LLMs) and show that prompting can be seen as means for improving the alignment at inference time (RQ3).

Our experiments reveal that: (i) When no distribution shift or utility drift happens, decoding based solely on the likelihood is enough to provide high utility, i.e., likelihood is a strong

predictor of utility. (ii) In such cases, there is no significant difference between different kinds of decoding algorithms, and we would recommend keeping beam search. (iii) In the presence of distribution shift or utility drift, value-guided beam search is both an effective and efficient decoding algorithm that leverages a value model at inference time to fix misalignment. Finally, (iv) for LLMs, prompting is a mechanism that sets the model in a state where the likelihood is well-aligned with the utility. This perspective provides a tentative explanation for the empirical success of prompting LLMs.

This work studies the fundamental problem in decoding, which involves a complex interaction between models, tasks, and data. Our unifying conceptual framework (the MMS taxonomy), accommodating all known decoding algorithms, enables the systematic study of decoding in a considerably broader scope than previously. By sharing the taxonomy and open-sourcing our implementations, we hope to pave the way for a more structured discussion in the future.

4.2 Background

High Utility Is the Goal. For a task t and input x, the utility function assigns a score $u_t(y|x)$ to each element y in the output space $\mathscr{Y}$. This score quantifies the goodness, or quality, of the output y with respect to a specific input x. For instance, in translation, the utility quantifies the extent to which the output conveys the same message as the input. For question answering, the utility simply quantifies the correctness of the answer. These task-specific notions of utility are operationalized in the evaluation metrics. The development of evaluation metrics that correlate with the human-defined notion of utility is a very active research area [217] and beyond the scope of this work. In our analysis, we use the canonical evaluation metric of each task as the utility function. For partially-decoded sequences, the utility can be approximated using a *value function* (see Sec. 4.3.3).

Given some input x, an *ideal model* would generate the element from the output space corresponding to the highest utility score: $\text{argmax}_{y \in \mathscr{Y}} u_t(y \mid x)$.

Unfortunately, most of the practically relevant utility functions are not amenable to optimization, forcing us to work with proxy functions, such as the canonical likelihood.

Language Models. A language model corresponds to a probability distribution p over $y \in \mathscr{Y}$, where $\mathscr{Y}$ is the set of all sequences that can be constructed using a vocabulary $\mathscr{V}$. In this work, we focus on conditional LMs $p(\cdot|x)$. Usually, these conditional distributions are modeled autoregressively (parametrized by θ): $p_\theta(y \mid x) = \prod_{i=1}^{|y|} p_\theta(y_i \mid y_{<i}, x)$. The model is trained to maximize the target sequence's conditional log-likelihood with teacher forcing, using the cross-entropy loss $\mathscr{L}(\theta) = -\sum_{(x,y)\in\mathscr{D}} \log p_\theta(y \mid x)$, where $\mathscr{D}$ is the training corpus [244, 245].

Once an LM is trained, it provides a next-token probability distribution across the output vocabulary. Decoding algorithms define how tokens are chosen during generation.

4.3 Proposed MMS Taxonomy

In this section, we propose a taxonomy of misalignment mitigation strategies (MMSs). This work focuses on decoding-based MMSs that mitigate the misalignment without modifying the model. As a primary signal, they rely on the model's likelihood. However, additional components (*e.g.*, value model, knowledge base, etc.) can be leveraged. Decoding algorithms, which we define as procedures that take in an input — and potentially some context (*e.g.*, a prompt with a task description or examples) — and return a sequence from the output space, can be seen as specific implementations of an MMS. Apart from fixing the misalignment problem at inference time, it is also possible to retrain or finetune the model with newly collected data that better reflect the intended utility and target testing distribution. We leave the detailed treatment of this part of the taxonomy for future work. See the last section in this chapter for further discussion of these alternatives.

4.3.1 Greedy Likelihood-Based Strategy

Given the LM's probabilistic formulation, one could strive to select the most likely sequence under the model: $\text{argmax}_{y \in \mathcal{Y}} p_\theta(y \mid x)$. However, due to the exponentially large state space, this optimization problem is intractable.

The class of algorithms following the greedy likelihood-based MMS approximate the intractable argmax by following the greedy heuristic of making *locally* optimal choices at each decoding step w.r.t. the *likelihood* under the language model. However, reaching a globally optimal solution may require locally sub-optimal steps. When this happens, we say that the likelihood landscape is *greedy adversarial* [160]. For a likelihood model that is not greedy adversarial, greedy heuristics will retrieve the highest-likelihood solution. Therefore, the algorithms' effectiveness depends on the likelihood–utility alignment.

Contrarily, greedy decoding algorithms may fall arbitrarily short of the global maximum for likelihood models that are greedy adversarial. Indeed, greedy decoding algorithms implicitly optimize a different objective function — a *tampered* version of the *likelihood* objective in which a term that encourages locally optimal solutions is added [160]. Therefore, the ability of greedy decoding algorithms to retrieve high-utility sequences even from a likelihood model that is perfectly aligned with the utility is inversely proportional to how greedy adversarial the likelihood landscape is. In some cases, the particular bias induced by the greedy heuristic *mitigates* the likelihood–utility *misalignment*, and makes the tampered likelihood objective better aligned with the utility than the original [160, 162, 241].

The decoding algorithms in this category can be further divided into two subgroups: (i) deterministic ones, such as greedy search (GS) and beam search (BS); and (ii) stochastic ones, such as top-k sampling [65], top-p sampling [102], and stochastic beams (SB) [129]. For more details, see Appendix B.1.1.

4.3.2 Greedy Likelihood-Based Strategy with Pruning

An understated fact in the literature is that even for tasks for which the canonical decoding algorithms (*e.g.*, BS) perform well, a non-negligible portion of the performance relies on some bespoke, ad-hoc tweaks on the likelihood scores [239]. These tweaks are usually based on either: (i) post-hoc observations that likelihood-based decoded sequences often contain specific undesirable patterns (*e.g.*, empty or short sequences, repetitive patterns, etc.); or (ii) problem-specific knowledge about the utility landscape suggests that high-utility sequences have a specific property, which can be explicitly enforced by the decoding strategy (*e.g.*, sequences should correspond to triplets of elements from a predefined set). Conceptually, all these tweaks employ mechanisms that *discourage* the generation of *high-likelihood* patterns that are known (or expected) to be associated with *low utility*.

This category includes: (i) decoding algorithms with ad-hoc heuristics such as the n-gram repetition penalty [126]; (ii) constrained beam search (CBS) [221, 54, 119]; (iii) NeuroLogic [155]. For more details, see Appendix B.1.2.

4.3.3 Greedy Likelihood- and Value-Based Strategy

Heuristics such as the ones used in the previous category can capture some properties associated with high utility but are limited to properties that can be easily expressed explicitly. While the utility function is generally defined for complete sequences only, to guide decoding algorithms, one can rely on the general concept of a value model. A value model approximates, for partially decoded sequences, the expected utility of the final sequence if the decoding keeps following the same policy. The most prominent algorithm in this category is value-guided beam search (VGBS) [95, 206, 130]. It uses a greedy strategy similar to BS but selects the next token using a linear combination of the LM's likelihood and the value model's scores. See Appendix B.1.3 for details.

4.3.4 Simulation-Based Strategy

Even though VGBS considers both likelihood and value, it remains greedy by only looking one step ahead. Simulation-based decoding algorithms explore further into the future before making the next decision. When the value landscape is complex and constructing a good value model is hard, such strategies with a look-ahead may become particularly effective. By turning the knob controlling the number of simulations, one can trade off computational efficiency for obtaining better value estimates. Monte-Carlo Tree Search (MCTS) is the canonical example of simulation-based tree exploration informed by a value model. For details, see Appendix B.1.4.

4.3.5 Prompting-Based Strategy

The decoding algorithms described in the last two sections address the likelihood–utility misalignment post-hoc. An alternative is to change the conditioning of the model's probability distribution such that the misalignment never happens. The effort now goes into choosing a context that aligns the likelihood landscape with the task-specific utility. The strength of this class of decoding algorithms lies in the fact that they can readily be applied to a new task without requiring any modifications to the model or increasing the computation cost of inference (beyond the processing of the prompts' tokens). However, they only work for large generalist LMs [42]. The most prominent members are the few-shot (FS) and the chain-of-thought (CoT) prompting methods, described in Appendix B.1.5.

4.4 Experimental Setup

4.4.1 Research Questions

In contrast to previous works that have studied the misalignment problem in constrained settings, we propose quantifying it in a unified and large-scale analysis across tasks (RQ1). We investigate the benefits of a diverse set of previously proposed solutions to the misalignment problem. Our study includes value-guided approaches (RQ2) and prompting (RQ3), covering each class of the MMS taxonomy with at least one representative. Specifically, we ask the following research questions.

RQ1: How correlated are the utility and the likelihood across tasks? As argued in Sec. 4.3, greedy likelihood-based strategies only require the likelihood to be a strong predictor of utility. We investigate whether this holds across tasks. Specifically, we measure two important aspects of the likelihood–utility alignment: (a) **Post-decoding alignment**. For each data point, the decoding algorithm chooses one output; we measure the likelihood and utility of the prediction and analyze their relation. Is high likelihood associated with high utility in the same way across tasks? (b) **During-decoding alignment**. Decoding algorithms typically explore a set of high-scoring candidates (e.g., BS returns one candidate per beam). We measure the correlation between likelihood and utility among these candidate outputs to analyze the likelihood landscape of the model.

Tasks	Utilities	Misalignment Types	Model	Dataset
Closed Information Extraction (cIE)	F1 score [M]	TI	GenIE (BART)	REBEL
Machine Translation (MT)	BLEU [M]	TI, DS	mBART50	WMT14
Non-Toxic Text Generation (NTTG)	Non-Toxicity [T]	TI, DS, UD	GPT2	RTP
Non-Soluble Protein Generation (NSPG)	Non-Solubility [T]	TI, DS, UD	ProtoGPT2	SwissProt
Sports Understanding	Solve Rate [M]	TI, DS, UD	MT-NLG 530B	Sports

Table 4.1: **Overview of the tasks.** Utility functions are categorized into: (a) [M]: Metric-based and (b) [T]: Trained model-based. The three misalignment types are TI: Training imperfection, DS: Distribution shift, UD: Utility drift.

RQ2: How effective are value-guided MMSs? In particular, we investigate the benefit of value-guided decoding algorithms as a function of the value model's quality.

RQ3: Is prompting an MMS? We investigate the efficacy of prompting as a likelihood–utility alignment tool for generalist LMs (LLMs).

4.4.2 Tasks and Datasets

To organize the discussion, we propose a simple classification of the sources of misalignment: (a) **Training imperfections (TI)**, when the model is trained on a different objective than the true utility, because of the finite size of the dataset and the approximation error in training (*e.g.*, via stochastic gradient descent); (b) **Distribution shift (DS)**, when the training and testing data distributions differ; (c) **Utility drift (UD)**, when the utility used in development differs from the utility at test time.

While we can expect TI to affect all modern neural models trained on large-scale datasets (due to the finiteness of datasets and approximations resulting from gradient-based training), DS and UD are task-specific. DS typically occurs when the distribution of the data changes between the training and testing scenarios. UD occurs when the notion of utility changes, *i.e.*, the labels for the same data points are changing.

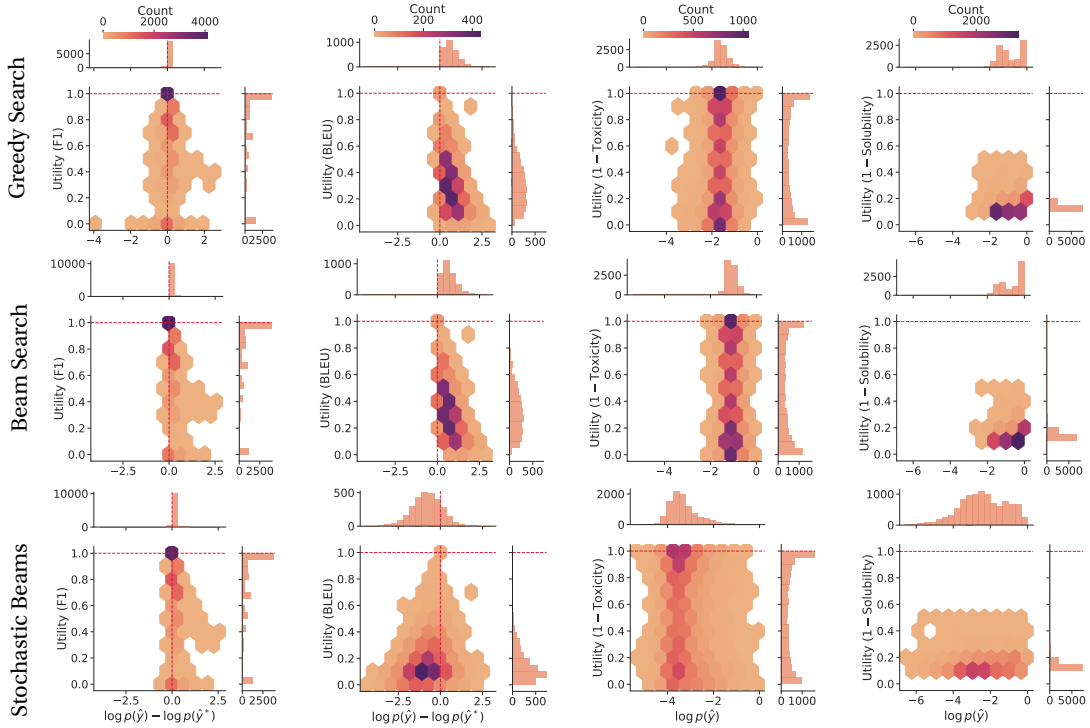
We carefully selected a variety of generation tasks covering (a) different *notions of utility*, and (b) different expected types of *(mis)alignment between utility and likelihood*. Table 4.1 gives a high-level overview of these tasks, their utility functions, and associated datasets.[1] In closed information extraction, the training and testing data come from the same distribution, and we expect only TI-type of misalignment. In machine translation, the mBART model is pretrained on a different dataset, inducing some DS as the texts used for training may come from different domains. For non-toxic text and non-soluble protein generation, the task definition changed from generating low perplexity sequences to generating non-toxic sequences. Therefore, UD is expected to be the main driver of misalignment. Similarly, for the sports understanding task, since MT-NLG was not trained for this specific task, we also expect UD to be the main source of misalignment, but here VGBS and MCTS are too expensive due to the size of the LM. Instead, we use this setting to investigate prompting-based MMSs.

4.4.3 Decoding Algorithms

To cover the full space of MMSs, we experiment with at least one representative from each class from the taxonomy defined in Sec. 4.3. From the Greedy Likelihood-based category, we include the canonical GS and BS, as well as the sampling-based SB. From the Greedy Likelihood-based Strategies with pruning, we use CBS. VGBS and MCTS are representatives of the Greedy Likelihood- and Value-based, and Simulation-based decoding classes, respectively.

[1] For more details about the models, data, and utility functions, see Appendix B.2.1.

For prompting, we consider the FS and CoT methods. The hyper-parameters for each algorithm are given in Appendix B.2.2. Appendix B.2.3, provides a complexity analysis in terms of LM and value model calls.



(a) cIE (REBEL) (b) MT (WMT14) (c) NTTG (RTP) (d) NSPG (SwissProt)

Figure 4.2: **RQ1 (post-decoding alignment)**: Each decoding algorithm is applied to each dataset-model pair. For each subplot, the x-axis represents the outputs' log-likelihood under the model, and the y-axis the output's task-specific utility score. The plots are frequency heatmaps counting the number of decoded outputs pertaining to a given hexagon. For MT and cIE, the x-axis is normalized such that 0 is the log-likelihood of the target answers[2]. These plots show where the outputs are located in the likelihood-utility landscape across tasks and decoding algorithms.

4.4.4 Value Models

The quality of a value model reflects its ability to approximate the expected utility. To determine the relationship between the value model's quality and the benefit of leveraging it in decoding, we craft models that allow us to instantiate versions with varying levels of quality, ranging from a random predictor to an oracle.

Non-Toxic Text Generation. The state-of-the-art method for detecting toxicity is via classification [94]. Such a classifier can readily be used as a value model in decoding. We reproduce the training procedure from Hanu and Unitary team [94] and save checkpoints at regular intervals

until the training is complete. Due to the gradually decreasing under-fitting, these checkpoints give us a sequence of classifiers that systematically improve in terms of quality.

Machine Translation. To achieve a similar effect for MT, we start by assigning to each data point in the dataset another randomly chosen data point which will serve as a false target. This assignment is fixed across all runs. During inference, the value model calculates the BLEU score for both correct and incorrect targets and returns a linear combination between the points. By gradually increasing the weight assigned to the false target from zero to one, the perfect value model (oracle) slowly degrades to a random predictor.

4.5 Experiments and Results

4.5.1 RQ1: The Likelihood–Utility Relationship

We present two analyses, one measuring the likelihood–utility misalignment after decoding and one measuring it during decoding.

Post-decoding alignment. In this experiment, we first run each likelihood-based decoding algorithm (GS, BS, and SB) for each dataset–model pair. Then, for each output, we compute both the model likelihood and the task-specific utility.[3] We report the results in Fig. 4.2.

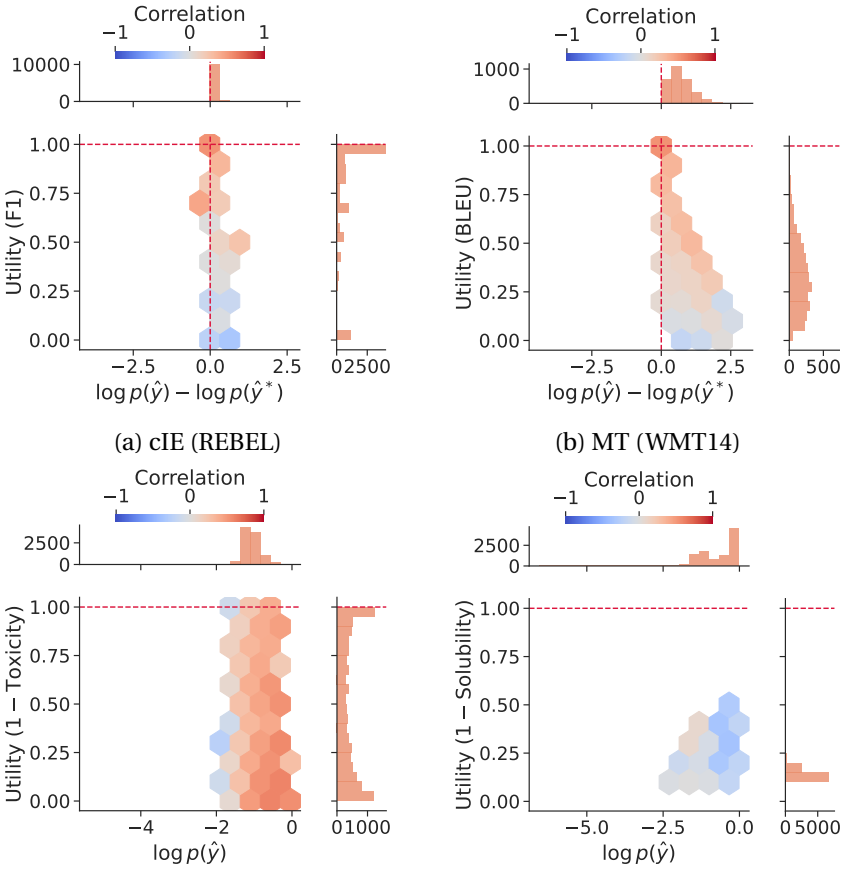
For cIE (first column), most outputs have a likelihood close to the targets' likelihood. The majority of the outputs have perfect utility — the decoded output is exactly the target, and the model is well-calibrated. This confirms the intuition that when the UD and DS are small, greedy likelihood-based MMS are very effective and can cope with the TI.

However, in tasks with larger DS and UD, the story is different. In MT (second column), the combined effect of TI and DS gives rise to a negative global correlation ($-.56$ for BS and $-.52$ for GS in terms of Pearson's correlation; $p < 10^{-3}$) between the predictions' utility and likelihood after decoding. This is an instance of *Goodhart's law*, where a surrogate metric (likelihood), when being optimized heavily, becomes a poor approximation of the original property it is supposed to track (utility).

In tasks with large UD, NTTG (third column), and NSPG (fourth column), decoding according to likelihood does not guarantee utility. For example, in NTTG, the likelihood–utility correlation is $-.10$ for GS, $.10$ for SB, and $.03$ for GS in terms of Pearson's correlation; $p < 10^{-3}$. These scenarios require external information that can guide the decoding towards high-utility outputs.

During-decoding alignment. Now, we investigate the likelihood–utility alignment where it matters: for outputs close to being extracted by the decoding algorithms. BS maintains k

[2] See Fig. B.1 for a plot without the target normalization.
[3] We also ran experiments with top-p and top-k but did not observe a behavior different from BS.

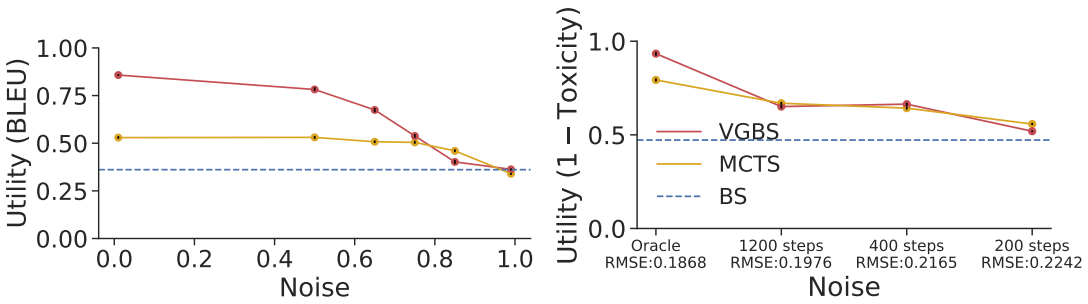


(a) cIE (REBEL) (b) MT (WMT14)

(c) NTTG (RTP) (d) NSPG (SwissProt)

Figure 4.3: **RQ1 (during-decoding alignment)**: For each dataset-model pair, we run BS and analyze the correlation between likelihood and utility of the top-5 candidate hypotheses. The y-axis represents the task-specific utility score, and the x-axis the log-likelihood under the model. The plots are generated as follows: (i) take the BS outputs from Fig. 4.2 with their log-likelihood and utility scores, which indicate the x and y coordinate of each data point; (ii) for each data point measure the Kendall's τ correlation between likelihood and utility of the top-5 candidate hypotheses; and (iii) average the correlation across the points belonging to the same hexagon.

candidate hypotheses, one per beam, before returning the top-scoring one as the final output. In this experiment, we analyze the correlation between the likelihood and utility of the top-5 candidates. The results are reported in Fig. 4.3. There are three dimensions to this problem: (i) the likelihood (x-axis); (ii) the utility (y-axis); (iii) the correlation (color). Ideally, we would like to see red everywhere, indicating that failure to retrieve a high-utility output is due to the decoding algorithm, but the likelihood of the model is still a good predictor of utility. However, this is not what we observe.



(a) MT (WMT14) (b) NTTG (RTP)

Figure 4.4: **RQ2**: For MT and NTTG, we ran VGBS and MCTS with value models displaying various levels of noise. We report the average utility of outputs on the y-axis (with 95% confidence interval). The noisy value models are described in Sec. 4.4.4.

For MT and cIE, we see a clear picture, red color (high likelihood–utility correlation) occurs at the top of the plot (high-utility): high likelihood-utility correlation among candidate outputs is enough to yield close to perfect-utility outputs.

For the NTTG (Fig. 4.3c), the correlation between utility and likelihood among the beams increases as the likelihood increases. When the model generates high-likelihood outputs, there is a positive correlation between being more likely and being less toxic. However, the likelihood mass is assigned to low-utility regions of the output space, which cannot be resolved with decoding based only on the likelihood. For NSPG (Fig. 4.3d), the correlation across all bins is negative, indicating that high likelihood is a very bad predictor of high utility.

Takeaways. When TI is the only cause of misalignment, the likelihood is a strong predictor of utility; then, likelihood-based decoding algorithms are expected to retrieve high-utility outputs. When UD and/or DS are present, the correlation between likelihood and utility post-decoding plummets, indicating that likelihood-based decoding algorithms are ill-suited. When UD is present (bottom row of Fig. 4.4), good correlation among the beams does not necessarily mean good utility. However, without UD (top row of Fig. 4.4), higher correlation among the beams is associated with high utility.

4.5.2 RQ2: The Benefits of Value Models

We now analyze the benefits of value-guided decoding algorithms (VGBS and MCTS) as a function of the value model's quality (see Sec. 4.4.4). Due to the high computational cost of running the experiments with both VGBS and MCTS, we focus on two tasks: MT and NTTG. For each version of the value model, we first perform a hyperparameter search on a small subset of the data and use the best hyperparameters on the test set. The results are reported in Fig. 4.4. VGBS and MCTS are always at least as good as BS, even with random value models, as the small-scale hyperparameter search selects parameters that ignore the values when they

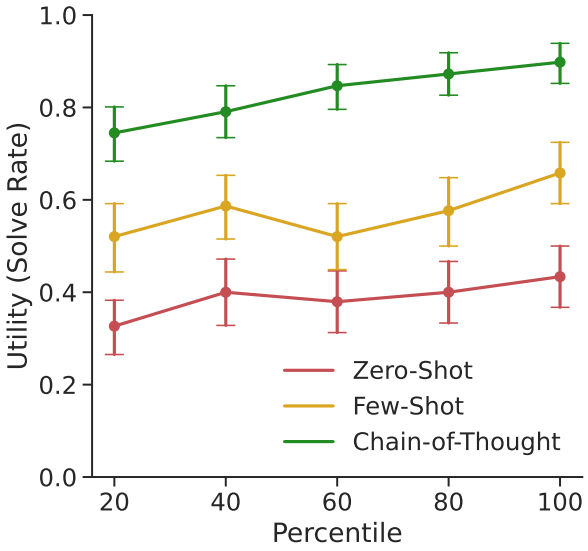


Figure 4.5: **RQ3**: ZS, FS, and CoT prompting, on the Sports understanding dataset. We report the utility (y-axis) of outputs binned according to the empirical percentiles of their likelihood (x-axis). The lineplot is the average utility per bin with 95% confidence intervals.

are not useful. However, when there is some signal in the value model, both VGBS and MCTS effectively leverage it and quickly start outperforming BS. When the value model is accurate, very high-utility outputs are discovered. Interestingly, VGBS mostly outperforms MCTS, and can extract almost perfect outputs, whereas MCTS plateaus. This is significant because VGBS has a substantially lower complexity than MCTS (see Appendix B.2.3).

Takeaways. Value-guided decoding algorithms can overcome the likelihood–utility misalignment and significantly outperform likelihood-based decoding algorithms even with noisy value models, as long as a small-scale hyperparameter search is done. VGBS offers a better trade-off between performance and computation cost than MCTS.

4.5.3 RQ3: Prompting as an MMS

Recently, Wei et al. [266] showed how the simple and broadly applicable idea of including in the prompt a few examples where the targets contain a sequence of steps that lead to the answer can greatly enhance the reasoning capabilities of LMs. We mimic the Sports Understanding task in their work by taking the same in-context examples and evaluating the same two prompting methods: CoT and standard FS prompting. Additionally, we evaluate the model in the standard zero-shot (ZS) setting, without any examples in the context, as a baseline. This results in unstructured answers that need to be labeled manually, see Appendix B.3.2.

To test our hypothesis that prompting is a means of addressing misalignment, we measure the utility and the likelihood under the model for all the testing data points. The results, summa-

rized in Fig. 4.5, provide three insights. First, similarly to Wei et al. [266], CoT outperforms FS and ZS with an accuracy of 83%, versus 57.3% and 38.9% for FS and ZS, respectively. Second, (and complementary to the information visualized on the plot) the average log-likelihood of the outputs generated by CoT is significantly higher than the FS and ZS generated outputs, -0.067 as opposed to -0.17 and -2.472. Third, the correlation between the likelihood and the utility when decoding with CoT is higher: 0.11 Pearson's correlation compared to 0.07 and 0.09 for FS and ZS, respectively.[4] Referring back to the observation made in RQ1b on Fig. 4.3, the value-guided MMSs studied in Sec. 4.5.2 address the misalignment post-hoc. However, the hidden representations building up to that misalignment are not modified, and the undesired information will still be attended to in predicting the next token probability distribution. In contrast, an effective prompting strategy addresses the misalignment before it affects the hidden representations, thereby (i) forcing the model to assign high likelihood to high-utility regions of the output space and (ii) improving the likelihood–utility alignment, making it easier to find high-utility outputs with greedy likelihood-based decoding algorithms.

Takeaways. Effective prompting methods put the model in a state where the generated outputs' likelihood is well-aligned with the desired utility.

4.6 Summary and Critical Discussion

RQ1 reveals that decoding based solely on the likelihood gives poor expected utility whenever DS or UD occurs. DS and UD make the likelihood a poor predictor of utility. When only TI is present, these decoding algorithms perform well because the likelihood is a strong predictor of utility.

By answering RQ2 and RQ3, we saw that methods bringing in external information at decoding time can effectively solve the likelihood–utility misalignment problem. While finetuning (or retraining) would be an obvious and apparently ideal MMS, this is often neither possible nor necessary. Indeed, our experiments show that if a value model can be crafted and we can afford the extra compute for the value model calls, then VGBS becomes a strong decoding algorithm capable of fixing misalignment problems at inference time. It is more efficient than MCTS and performs better than BS, even if the value model is only a poor approximator of the utility. When crafting a useful value model is difficult (e.g., protein function depends on the 3D structure, which cannot be easily approximated from partial amino-acid sequences), MCTS with a large number of simulations with roll-outs can be used to "estimate" one. However, the price to pay is a higher computational cost at inference time. Finally, for large, generalist LMs, decoding algorithms such as MCTS or VGBS are prohibitively expensive due to the high computational cost of each call to the LM. Prompting methods combined with greedy or top-p decoding can be considered as a way to leverage external information in the form of few-shots prompts to set the model in a state where the likelihood is better aligned with the utility. Our

[4] The differences are statistically significant ($p < 10^{-3}$).

experiments support this explanation of the success of prompting large LMs. For a similar perspective, comparing prompting methods with training-based MMSs see He et al. [96].

Non-exhaustive empirical analysis. This work studies a fundamental problem that involves a complex interaction between tasks, models, and data; and sequence-to-sequence models have been applied to a very broad set of tasks. Covering all possible combinations is impossible, and for our empirical analysis, we chose a subset to evaluate. Our choice is guided by the classification of misalignment sources proposed in Sec. 4.4.2 and aims to cover different areas of the misalignment space. A seemingly small difference between two choices (e.g., a difference in the loss function used in training the model) can give rise to a considerably different misalignment and, consequently, performance. This is why the goal of the proposed conceptual framework is to make a step toward enabling a more systematic study of decoding. To further help the community investigate the broader space of tasks, models, and datasets through this lens, we open-source the implementation of our analysis.

Alternative ways of fixing misalignment. Apart from value-based decoding, other techniques could be considered to fix the misalignment problem: (a) Retrain or finetune the model with data that better reflects the task's utility. For instance, to generate non-toxic text, one could retrain or finetune the language model on curated datasets that contain toxic prompts and non-toxic sentence continuations. (b) Optimize more directly the utility function instead of surrogate differentiable objectives. This could be done via reinforcement learning (see Wang, Li, and He [261] and Wu et al. [277] for BLEU). In this work, we focused on decoding algorithms and ways of fixing the likelihood–utility misalignment problem at inference time. Future research could further investigate the trade-offs involved in finetuning and retraining. Is it better to invest resources in acquiring new data that fits the task for finetuning? Or is it better to fix DS and UD at inference time with VGBS, MCTS, or prompt engineering? Where do the inflection points lie?

Part III Training LLMs to Be Collaborative

Preface

In Part II, we show how decoding algorithms can serve as an efficient strategy for integrating an LLM in a collaboration without modifying the underlying model. Naturally, for some tasks, this approach may not be sufficient, necessitating fine-tuning. A challenge arises when the training signal required for such improvement is not readily available.

One promising approach for addressing this issue is the synthetic generation of training data. In Chapter 5, we study the question of *how to synthetically generate data even for tasks that the LLM cannot solve directly*. The chapter concludes by drawing connections between the proposed ideas and work on self-improvement for LLMs, highlighting the broader implications of our work.

5 Asymmetry for Synthetic Data Generation

5.1 Introduction

Prompting large language models (LLMs) has been a reliable technique for achieving impressive performance across domains [24, 268, 127]. An important application of this capability is the generation of large amounts of high-quality synthetic data for training and evaluating smaller models. This becomes particularly useful for tasks where high-quality datasets are not readily available or access to real data is limited or expensive. However, in many complex natural language processing (NLP) tasks, the textual input x is mapped to a particularly *structured* (rather than free-text) output y, and in such cases, LLMs may perform poorly as synthetic-data generators, since pretraining did not gear them to produce the specific required output format (even with in-context learning). Here, we propose to alleviate this issue by generating synthetic data in the reverse direction by first sampling an output structure y and then prompting the LLM to generate a corresponding input text x (see Fig. 5.1). We postulate that for many tasks, given appropriate in-context information, an LLM can generate a meaningful x corresponding to an output y, even when the original task cannot be solved directly by the LLM. Exploiting this asymmetry, then, will allow us to synthetically generate high-quality data even for hard tasks. Furthermore, the flexibility to choose the distribution over output structures y gives us much-desired fine-grained control over the data distribution.

A good example of such a task, on which we focus, is *closed information extraction* (cIE)—the task studied in Chapter 3. In cIE, a model must extract a set of disambiguated triplets (*i.e.*, facts) y from a given text x. These triplets are of the format (subject, relation, object), with the subject and object being entities in a knowledge base (*e.g.*, Wikidata) and the relation being specified in the knowledge base schema.

Collecting datasets for this task is time-consuming and expensive, as it requires annotators to know the entire entity and relation catalogs and reason about all possible facts expressed in the text x. As a result, only small or noisy datasets exist to date. In particular, Sec. 3.5.2 shows that the largest available dataset, REBEL [111], suffers from several problems : (i) Noise: it is collected with a mix of heuristics, and for many data points, the target output y does not

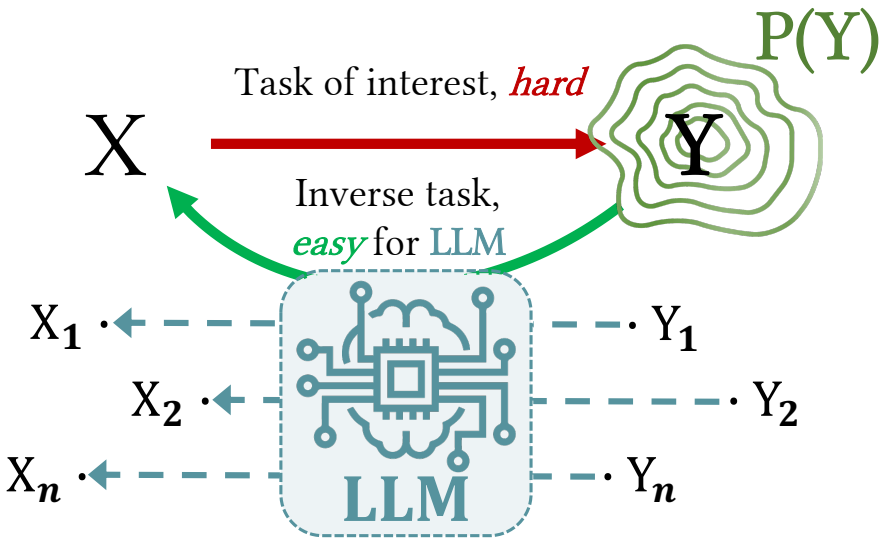


Figure 5.1: **Exploiting asymmetry for SDG.** For hard tasks of interest with input X and output Y, the reverse task (from Y to X) may be much easier for an LLM. If so, we can generate high-quality training pairs (X, Y) by prompting an LLM to generate plausible inputs X from outputs Y. This often holds true for tasks with structured Y, as in closed information extraction, where X would be the input text and Y would be the list of (subject, relation, object) triplets expressed in the input text. Furthermore, this ensures full control over the sampling distribution $P(Y)$, and thus balanced datasets.

contain all the facts expressed in the input x or is (partially) incorrect. (ii) Skewness: most relations appear only rarely in the dataset, which results in models that ignore most of the information when the data is used for training and in poor performance estimates when it is used for evaluation.

Importantly, cIE is also not solvable directly by LLMs such as GPT3.5, as such models are unaware of the entity and relation identifiers of interest (for examples of failures, see Appendix C.1). We show that, although the LLM cannot be used to produce output structures directly, we can use it to generate high-quality input texts by reversing the task.

Contributions. (i) We propose a *strategy* for designing effective *synthetic data generation (SDG) pipelines* and apply it to cIE. Concretely, we start by sampling triplet sets from the Wikidata knowledge graph (KG) such that each triplet set is coherent (*i.e.*, can be expressed by a short text) and the selected triplet sets cover all relations more uniformly than REBEL does. For each triplet set y, we then prompt an LLM to generate text x that expresses these, and only these, triplets. This results in a high-quality synthetic dataset (1.8M data points) that we use to replace the noisy REBEL dataset. The process is illustrated in Fig. 5.2. (ii) In a human evaluation, we show that our synthetically generated data is of significantly *higher quality* than the existing REBEL dataset, has a *more uniform relation-frequency distribution,* and can

be *generated cheaply* at scale. The evaluation reveals that, with 70% of the information from the text not present in the target set and 45% of the target triplets not actually expressed in the text, REBEL's test set cannot be used to accurately estimate performance. In contrast, for our highest-quality test set, the corresponding numbers are 15% and 22%, respectively. (iii) We introduce *SynthIE,* a series of Flan-T5 models [43] finetuned on our synthetic data. In contrast to previous models, which perform well only for the few most frequently occurring relations, SynthIE achieves high precision and recall across all relations. On REBEL Clean, a manually annotated subset of REBEL, SynthIE's macro-F1 score even exceeds that of the original REBEL data's gold annotations. On Wiki-cIE Text, our highest-quality test set, SynthIE outperforms the previous state of the art (with equal model size) by a substantial margin of 57 and 79 absolute points in micro-F1 and macro-F1, respectively.

Overall, we demonstrate that by exploiting asymmetry between structured outputs y and textual inputs x, LLMs can generate high-quality synthetic data to train smaller, specialized models. This way, a task that was previously not solvable by LLMs is now feasible even for a small 220M-parameter model. Our code, models, and datasets are released for future researchers to reuse and extend.

5.2 Background and Related Work

5.2.1 Synthetic Data Generation

Several approaches to data augmentation rely on pretrained language models (PLMs). Early efforts include works that require the pre-existence of a dataset, which is then used to finetune the pretrained generator network [4, 188, 285, 175, 131].

Recent work focuses on methods that do not require supervision beyond a few demonstrations. Wang et al. [265] generate synthetic labels by providing unlabeled samples as examples to the LLM. Ye et al. [291] and Gao et al. [77] use PLMs to generate data with carefully designed prompts. They evaluate on text classification, question answering, and natural language inference. There are similar procedures for GLUE [260] tasks [165], intent classification [216], and question answering [143]. Alternatively, Meng et al. [167] tune a PLM on a few demonstrations and use it as a synthetic data generator; and Smith et al. [235] incorporate prompted PLM for weak-supervision; while Shao et al. [227] generate synthetic demonstrations to improve the prompting of LLMs; and Honovich et al. [104] generate synthetic instructions for instruction tuning of LLMs. Contrary to these works, we do not generate labels but prompt an LLM to perform the reverse task when the reverse task is *easy* for the LLM, which results in high-quality data points. To the best of our knowledge, only Meng et al. [166] and Gao, Fisch, and Chen [79] also perform the reverse task by prompting an LLM to generate comments x given a sentiment y. However, their direct task is simple and can already be solved easily by an LLM, and their y is not structured.

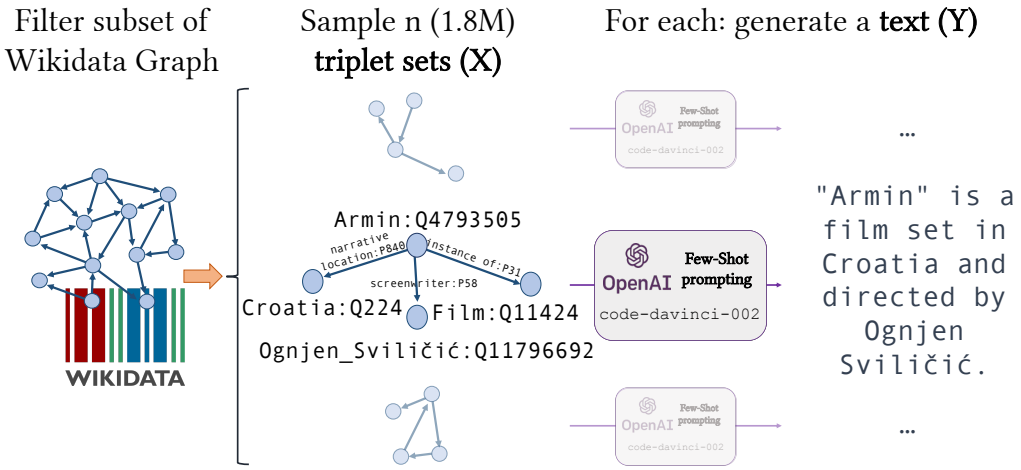


Figure 5.2: **Synthetic data generation flow.** We start by filtering the Wikidata knowledge graph to a subset of relations and entities comparable to REBEL. Then, we sample coherent triplet sets, encouraging uniform coverage of relations. Finally, we prompt OpenAI LLMs to generate text for each triplet set.

5.2.2 Data as a Core Limitation

As shown in Chapter 3, cIE models trained on existing datasets struggle with heavily skewed relation distributions. While our proposed approach, GenIE, achieved state-of-the-art performance with a 3x improvement over the strongest baseline, its macro-F1 score remained below 35%. This limitation stems primarily from the inherent data distribution, where most relations appear only a few times in the training data, leading to models that perform well only on frequent relations while struggling with rare ones.

Addressing this issue requires access to more annotated data covering the large number of rare relations that cannot be easily obtained through traditional methods like human annotations or distant supervision heuristics [111]. Instead, our synthetic data generation procedure offers a solution by enabling the generation of large-scale training data with comprehensive coverage across all relations.

The data scarcity problem is not unique to cIE, and our procedure can benefit other structured NLP tasks such as entity linking, oIE, or abstract meaning representation parsing [14]. This challenge extends beyond traditional NLP tasks to fundamental LLM capabilities, as high-quality training data for complex behaviors, like detailed reasoning traces, is scarce in naturally occurring web text.

5.3 Exploiting Asymmetry for Synthetic Data Generation

In this section, we demonstrate how to exploit asymmetry for synthetic data generation with cIE as an example task. Our pipeline, depicted in Fig. 5.2, comprises three primary

components: (i) construction of a knowledge graph (KG) containing the entities and relations of interest; (ii) sampling of coherent triplet sets from the KG with comprehensive coverage of the entities and relations, and (iii) generation of high-quality text, expressing the triplets without any additional triplets. Next, we describe these three components in turn.

5.3.1 Knowledge Graph Construction

We start from the Wikidata KG [259], and to remain comparable with Chapter 3, we filter the KG to the subset of 2.7M entities $\mathscr{E}$ and 888 relations $\mathscr{R}$ appearing at least once in REBEL's training set. Each entity in the KG is associated with a unique English Wikipedia page title, and each relation is linked to a unique Wikidata label, which we use as their textual identifiers (see Appendix C.2.1 for details).

5.3.2 Sampling Triplet Sets

In the Wikidata KG, nodes represent entities, and edges represent relations between two entities. Therefore, a triplet can be seen as an edge together with its endpoint nodes. For the synthetic data to be valuable, we design the triplet set sampling with two objectives in mind. First, it is crucial that the triplet sets be coherent: the triplets in each set must be able to conceivably co-occur in human-written text. Second, the dataset should have (approximately) uniform coverage of entities and relations.

Encouraging coherence. We observed that uniform edge sampling from the KG does not produce coherent triplet sets. Instead, coherent sets of triplets tend to focus on a small number of entities, with one or two entities being repeated in most of the triplets. These entities serve as the *"protagonists"*, or anchors, of the sentence. To capture this property, we propose a sampling procedure based on a random walk with backtracking applied to the KG. Concretely, given a starting point—a node (*i.e.*, an entity $e^{(0)}_{\text{sub}}$) or an edge (*i.e.*, a triplet $t_0 = (e^{(0)}_{\text{sub}}, r^{(0)}, e^{(0)}_{\text{obj}})$) from the KG—we maintain a set of already sampled triplets T, and until the desired number of triplets is reached, iteratively sample: (i) a subject $e^{(|T|)}_{\text{sub}} \in \mathscr{E}$ starting a new triplet; or (ii) an object $e^{(|T|)}_{\text{obj}} \in N(e^{(|T|)}_{\text{sub}})$, where $N(e)$ corresponds to the set of entities adjacent to e, forming a triplet $t_{|T|} = (e^{(|T|)}_{\text{sub}}, r^{(|T|)}, e^{(|T|)}_{\text{obj}})$ to be added to T. The entity sampling is biased towards entities already appearing as a subject or an object of a triplet in T by a parameter controlling the strength of the bias. The desired number of triplets per set is sampled from a Poisson distribution. Appendix C.2.2 details the choice of parameters.

Encouraging coverage. When sampling triplet sets from the graph uniformly, some entities and relations are so central that they appear in most local neighborhoods. These end up being over-represented, heavily skewing the distribution.

To alleviate this issue, we implement an aggressive reweighting of the entity and relation distribution. After every K sampled sets, we craft new relation and entity distributions, where

the probability of sampling a specific entity or relation is inversely proportional to its frequency in the set of already sampled triplet sets S. As a consequence, after each reweighting, the rarest entities and relations are given the highest probability of being sampled. We denote the two distributions as $\mathbb{D}^S_{\mathscr{E}}$ and $\mathbb{D}^S_{\mathscr{R}}$. Appendix C.2.2 details the choice of K.

Ensuring coverage. Rare entities and relations do not often appear in other entities' local neighborhoods. Even if they are associated with a high probability of being sampled when encountered, they are encountered rarely and, therefore, sampled rarely. One way to ensure that rare entities and relations are selected is to explicitly choose them as a starting point of a random walk (*i.e.*, a triplet sets). We describe two strategies for achieving this:
(i) **Entity-centric** strategy: sample the starting entities of each triplet set according to the reweighted entity sampling distribution $\mathbb{D}^S_{\mathscr{E}}$.
(ii) **Relation-centric** strategy: sample a relation r according to the reweighted relation sampling distribution $\mathbb{D}^S_{\mathscr{R}}$. Then, among the triplets corresponding to relation r, sample one according to the probability assigned to the subject entities by the reweighted entity sampling distribution $\mathbb{D}^S_{\mathscr{E}}$, renormalized to the ones available.

In both cases, reweighting favors the rarest entities and relations as the next starting points.

For comprehensive coverage of both entities and relations, we employ a **mixed** strategy, which switches between the entity-based and relation-based strategy on every K samples—at the same time when the empirical relation and entity distributions are recomputed.

5.3.3 Triplet-Set-to-Text Generation

In this section, we denote by *query* the triplet set for which we want to generate text. Also, we refer to the in-context examples as *demonstrations*. The demonstrations consist of triplet sets and sentences selected from REBEL's training set. When choosing the triplet-set-to-text generation setup, we made the following considerations.

LLM choice. We consider two models from OpenAI's GPT 3.5 series: `code-davinci-002` and `text-davinci-003` (details in Appendix C.2.3).

Prompting strategy. We evaluated both models in a zero-shot and a few-shot setting. In the zero-shot setting, we experimented with different instructions. In the few-shot setting, we varied the instruction, the number of demonstrations, and the formatting of the demonstrations.

Generation parameters. We experimented with different values for temperature and top-p.

We ran the inference on a selected set of 12 triplet sets from REBEL's validation set and manually evaluated the quality of the generated outputs in terms of precision, recall, and fluency. The best-performing prompt setup and optimal set of generation parameters for both models are given in Fig. C.2 and Table C.1, respectively. In Table C.2, we showcase example generations

	min	1st quartile	median	3rd quartile	max
REBEL	1	4	34	432	716,679
Wiki-cIE Code	65	934	1380	3629	479,250
Wiki-cIE Text	4	42	62	136	14,323

Table 5.1: **Relation occurrence count statistics.**

	Micro			*Macro*
	Precision	Recall	F1	Recall
REBEL	29.35 ± 7.77	56.05 ± 10.40	39.87 ± 7.62	24.20 ± 6.20
Wiki-cIE Code	57.40 ± 10.28	70.38 ± 7.83	65.08 ± 7.35	50.70 ± 9.10
Wiki-cIE Text	84.78 ± 5.80	78.45 ± 8.20	82.97 ± 5.53	72.14 ± 8.73

Table 5.2: **SDG quality (human evaluation) results.**

for a few data points. In this setup, we generated two datasets:
Wiki-cIE Code consists of around 1.8M training, 10K validation, and 50K test samples generated with `code-davinci-002`.
Wiki-cIE Text consists of 10K validation and 50K test samples generated with `text-davinci-003` using the same triplet sets as in Wiki-cIE Code.
Appendix C.2.3 contains the inference costs details.

5.3.4 Distributional Properties of Data

One important problem we aimed to address with our synthetic data generation is the imbalance in relation frequencies. Table 5.1 reports the basic statistics of the relation frequency distribution in REBEL, Wiki-cIE Code and Wiki-cIE Text (see Appendix C.2.4 for the cumulative distribution plots). While REBEL is very skewed towards few relations appearing most of the time, Wiki-cIE Code has a much more balanced distribution. In particular, the rarest relations in Wiki-cIE Code appears more often than the median relation in REBEL. Sec. 3.5.2 empirically shows that supervised models perform poorly on rare relations. Therefore, we expect Wiki-cIE Code to help supervised models perform well on a much larger subset of relations, which is necessary for exhaustive extraction. In terms of entity coverage, the training split of our Wiki-cIE Code contains 1,805,504 unique entities, compared to the 1,715,922 in REBEL's.

5.3.5 Human Evaluation

Experimental setup. We randomly selected 50 data points from REBEL's test set and synthetically generated the text for the corresponding triplet sets following the procedure outlined in Sec. 5.3.3, with the generation parameters and prompts used to generate the Wiki-cIE Code and Wiki-cIE Text datasets. As a result, two more versions of the (small) dataset, differing only in the

textual sequences, were created—one following the Wiki-cIE Code and another following the Wiki-cIE Text text generation procedure. We evaluate the match between the triplet-set-to-text pairs for each dataset by scoring data points in terms of standard precision, recall, and F1 (see Appendix A.3 for definitions). Concretely, we extract the triplets actually expressed in each of the three versions of the text by human annotation and compare them against the corresponding target set (*i.e.*, REBEL's gold triplet set) as ground truth. Appendix C.2.5 details this process. In this setting, precision corresponds to the fraction of triplets expressed in the text that are present in the target set, and recall to the proportion of triplets in the target set that were expressed in the text.

Results. The results are summarized in Table 5.2. First, the results indicate that the synthetically generated text has substantially higher precision and recall than REBEL's original text, with Wiki-cIE Text reaching 84.8% precision and 78.5% and 72.1% in micro- and macro-recall, respectively. Second, REBEL texts score low in precision (29.4%), suggesting that over 70% of the information in REBEL text is absent from the target set. On the other hand, a micro-recall score of 56%, implies that REBEL's gold annotations are actually wrong 44% of the time. Finally, our datasets' micro and macro scores are much closer than REBEL's, indicating that our datasets have more consistent quality across relations.

5.4 Synthetic Data in Action

In this section, we evaluate the benefits of training on synthetically generated data.

5.4.1 Modeling and Inference

Model. Given textual input x, our proposed model, SynthIE, autoregressively generates the linearized sequence representation y of the exhaustive set of facts y_{set} expressed in x. The conditional probability (parametrized by θ) assigned to the target set y_{set} is computed as: $p_\theta(y \mid x) = \prod_{i=1}^{|y|} p_\theta(y_i \mid y_{<i}, x)$. The training consists of maximizing the target sequence's conditional log-likelihood with teacher forcing [244, 245], using the cross-entropy loss, and dropout [238] and label smoothing for regularization [246]. While we maintain the same task formulation and training as GenIE, which employs the BART architecture [140], SynthIE is based on FLAN-T5 [43]—a choice mainly motivated by the availability of pre-trained checkpoints with different parameter counts.

Output linearization. To represent the set of facts y_{set} as a sequence of symbols y compatible with sequence-to-sequence architectures, we introduce two mappings: (i) *fully expanded* (FE) and (ii) *subject-collapsed* (SC), which was used by Huguet Cabot and Navigli [111]. While FE concatenates the textual representation of each triplet in the set to obtain the sequence, SC groups the triplets based on the subject entity and concatenates their grouped representations. For more details and examples, see Appendix C.4.

Inference. At inference time, it would be prohibitively expensive to score every set of triplets in the output space. Instead, we search for the top-k eligible options using constrained beam search, as in Chapter 3, paired with a strategy for dynamically generating the valid prefixes. More concretely, we enforce a bi-level constraint where (i) the high-level structural constraint asserts that the prefix follows a specific linearization schema, and (ii) lower-level validity constraints (via a pre-computed entity and relation trie) ensure only valid entity or relation identifiers (depending on the given element) are generated.

5.4.2 Experimental Setup

Knowledge base constraints. We maintain our world of concern to all the entities and relations from the KG described in Sec. 5.3.1 corresponding to labels that can be fully tokenized by the model's tokenizer (*i.e.*, tokenized labels do not contain unknown tokens), which rules out around 5% of the entities in the KG. Our final entity catalog contains around 2.6M, and the relation catalog 888 items.

Datasets. We differentiate between two data regimes: (i) synthetic and (ii) non-synthetic (distantly supervised). For the synthetic regime, we leverage the datasets generated by the SDG procedure described in Sec. 5.3. Concretely, we use the larger Wiki-cIE Code for training and testing and the smaller Wiki-cIE Text for testing purposes only. For the non-synthetic regime, we follow the same evaluation setup as in Chapter 3 (see Sec. 3.4.2 for details).

Baselines. To isolate the effect of training on synthetic data, we keep the same architecture and vary the training (and validation) data. We use the *GenIE* identifier to refer to models trained in the non-synthetic and *SynthIE* to refer to models trained in the synthetic regime. See Appendix C.3 for details on the hyper-parameters and the compute time.

Evaluation metrics. We evaluate the performance in terms of micro and macro precision, recall, and F1, and report a point estimate with a 95% confidence interval constructed from 50 bootstrap samples. Appendix A.3 formally describes the metrics.

5.4.3 Results

Human Evaluation on REBEL

The human evaluation in Sec. 5.3 uncovers fundamental flaws in REBEL's annotations. Approximately 70% of the information from the text is not included in the "gold" set of triplets, and 45% of the triplets are not expressed in the input text (*cf.* Table 5.2). As a result, evaluation on REBEL would provide a highly inaccurate picture of the models' performance. Specifically, triplets missing from the "gold" set lead to an underestimation of true precision, while incorrect triplets in the "gold" set can result in an overestimation of precision and an underestimation of recall. These findings raise serious doubts about the validity of REBEL as an

	Distant Supervision			*Synthetically Generated*					
	REBEL Clean			Wiki-cIE Text			Wiki-cIE Code		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Micro									
REBEL Gold	92.71 ± 1.73	60.68 ± 2.85	73.76 ± 2.20	–	–	–	–	–	–
GenIE T5-base	76.06 ± 3.42	51.81 ± 3.44	62.17 ± 3.01	49.10 ± 0.33	26.69 ± 0.17	34.58 ± 0.20	41.56 ± 0.49	23.94 ± 0.24	30.38 ± 0.30
SynthIE T5-base	53.02 ± 5.00	43.20 ± 3.06	48.05 ± 3.41	92.08 ± 0.17	90.75 ± 0.21	91.41 ± 0.18	79.99 ± 0.29	70.47 ± 0.30	74.93 ± 0.27
SynthIE T5-base-SC	59.97 ± 4.34	30.54 ± 2.14	40.76 ± 2.57	92.79 ± 0.12	90.50 ± 0.10	91.63 ± 0.10	81.58 ± 0.15	69.48 ± 0.29	75.05 ± 0.19
SynthIE T5-large	68.25 ± 4.91	54.37 ± 3.08	61.26 ± 3.07	93.38 ± 0.11	92.69 ± 0.19	93.04 ± 0.13	82.60 ± 0.19	73.15 ± 0.29	77.59 ± 0.24
Macro									
REBEL Gold	51.21 ± 5.03	41.02 ± 4.69	43.76 ± 4.62	–	–	–	–	–	–
GenIE T5-base	39.36 ± 4.68	31.46 ± 4.24	33.33 ± 4.07	29.82 ± 0.67	11.14 ± 0.15	13.94 ± 0.17	25.78 ± 0.85	9.81 ± 0.10	12.12 ± 0.12
SynthIE T5-base	35.57 ± 4.82	34.05 ± 4.47	33.13 ± 4.44	94.10 ± 0.15	92.42 ± 0.17	93.05 ± 0.11	83.76 ± 0.36	74.05 ± 0.45	77.91 ± 0.42
SynthIE T5-base-SC	20.07 ± 3.26	12.82 ± 2.65	14.65 ± 2.70	94.35 ± 0.19	92.39 ± 0.20	93.15 ± 0.15	84.32 ± 0.32	73.57 ± 0.41	77.88 ± 0.34
SynthIE T5-large	54.11 ± 5.26	52.01 ± 4.64	51.04 ± 4.76	95.27 ± 0.22	94.95 ± 0.13	94.99 ± 0.12	86.43 ± 0.25	78.78 ± 0.27	81.95 ± 0.22

Table 5.3: **Main results.** Performance of our model SynthIE, the baseline GenIE, and REBEL's target annotations, evaluated on the hand-annotated but biased REBEL Clean, Wiki-cIE Code, and the highest-quality Wiki-cIE Text.

evaluation dataset for cIE. Both of these problems (missing and extra triplets) are addressed by our proposed evaluation datasets (*cf.* Table 5.2).

To understand the implications of these limitations, we manually annotate 360 randomly selected samples from REBEL. This process results in a new dataset that we refer to as REBEL Clean. We provide detailed information about the annotation procedure in Appendix C.5.1.

We first evaluate REBEL's gold triplet sets against the hand-annotated triplet sets, by treating the original ones as if they were the output of a model (referred to as REBEL Gold in Table 5.3). The original annotations achieve an F1 score of 73.8 micro and 43.76 macro, which is unsatisfactory for a dataset intended for estimating model performance. Furthermore, the significant gap between micro and macro scores confirms that the quality of original annotations varies greatly across relations.

Next, we evaluate the predictions from our models and the baseline, and observe that, in terms of macro performance, (i) SynthIE T5-large outperforms REBEL Gold; and (ii) SynthIE T5-base is on par with GenIE T5-base. Crucially, the first observation suggests that the predictions of SynthIE T5-large—a model trained on synthetic data generated with our proposed methodology—exhibit higher quality than REBEL's original gold triplet sets. On one hand, this highlights the quality of SynthIE, and on the other hand, it further undermines the credibility of an evaluation on REBEL.

It is worth noting that the REBEL Clean dataset inherits some important problems from REBEL: (i) a high imbalance in terms of the relation occurrence counts, which can be exploited by models like GenIE (trained on REBEL) to achieve strong (micro) performance despite only performing well for few relations, (ii) text often containing information for entities that cannot be resolved.

These findings emphasize the importance of the proposed Wiki-cIE Text as a reliable evaluation dataset for the cIE task. By design, Wiki-cIE Text does not suffer from these issues.

Performance Evaluation

On Table 5.3, we start by noticing that GenIE T5-base achieves an F1 score of 62.2 micro and 33.33 macro on REBEL Clean. However, the model's performance decreases by almost half in terms of micro and two-thirds in macro F1 on Wiki-cIE Text and Wiki-cIE Code. This is due to several reasons. Crucially, the annotations per relation in GenIE's training dataset, REBEL, are not uniform in terms of representation and quality. As a result, the model performs well on a few relations and badly on the rest. This is exposed by the synthetic datasets, which (i) contain triplets expressing every relation seen at least once in REBEL's training set as opposed to REBEL's test set, which does not cover 300 relations (around a third); and (ii) express all of the relations as uniformly as possible.

Additionally, while GenIE's precision also decreases, F1 performance is particularly affected by the strong decrease in recall. For instance, on Wiki-cIE Text, in comparison to REBEL Clean, the precision drops by 17 (micro) and 10 (macro) absolute points while the recall drops by 25 (micro) and 20 (macro) absolute points. The drop in recall is more pronounced as a consequence of (i) training on an imbalanced dataset (REBEL), with non-exhaustive annotations missing 70% of the information; and (ii) evaluation on a balanced, diverse dataset in which almost 85% of the information from the text is present in the target triplet set (see Sec. 5.3.5).

On the other hand, SynthIE T5-base, which is trained on data synthetically generated by the proposed methodology, and differs from GenIE T5-base only in terms of the training data, achieves a 91.4 micro, and an even higher 93.1 macro-F1 score on Wiki-cIE Text. Going to the larger SynthIE T5-large model, the performance on both datasets increases from 2 to 4 absolute points.

Finally, the subject-collapsed (SC) linearization decreases the target sequence length (see Fig. C.5) without any performance costs on Wiki-cIE Text and Wiki-cIE Code. However, the autoregressive nature of the model, paired with the often ill-defined nature of the task in REBEL, renders SC an inappropriate choice for REBEL Clean that comes with a substantial performance cost. This result highlights that the choice of the output format should not be neglected. We further discuss this in Sec. 5.5.

Performance by Relation Frequency

There is a natural imbalance of relation frequencies in text. In existing datasets, most of the triplets correspond to only a few relations (see Sec. 3.5.2). Models trained on such data are good at extracting information concerning a few relations and ignore the rest, which is a major obstacle to exhaustive cIE. For this reason, we bucket relations based on their number of occurrences in REBEL's training set and compute the per-bucket (micro) F1 performance. The

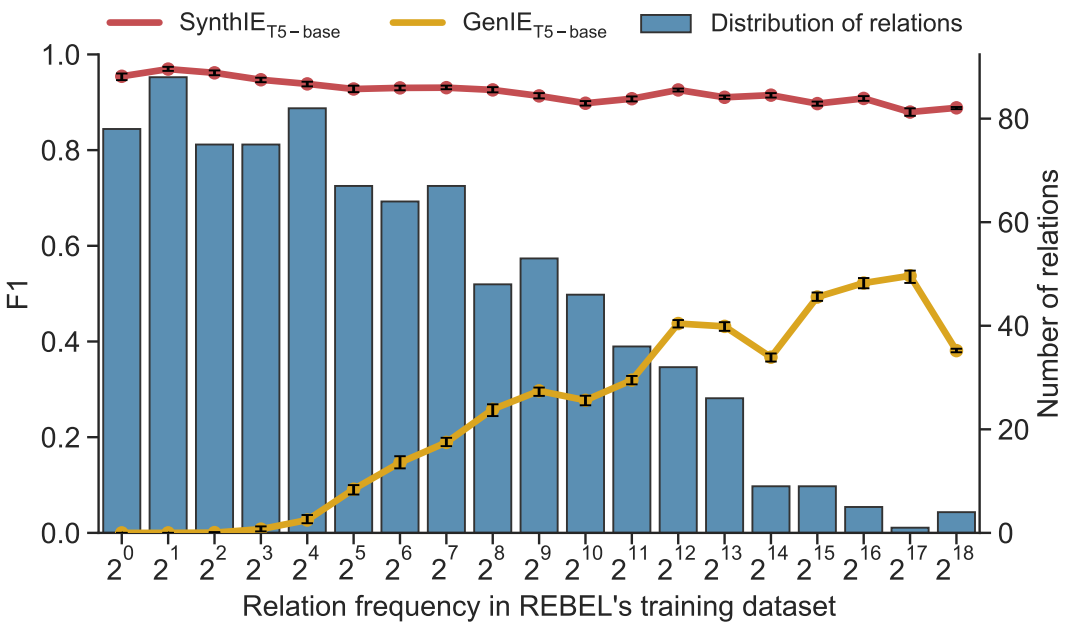


Figure 5.3: **Impact of the relation frequency.** Relations are bucketed based on their frequency; bucket 2^i contains relations occurring between 2^i and 2^{i+1} times. The histogram shows the number of relations per bucket. The line plots depict the per bucket F1 scores for GenIE and SynthIE evaluated on Wiki-cIE Text, with confidence intervals constructed by bootstrapping.

results are reported in Fig. 5.3. For 46% of the relations that have less than $2^5 = 32$ occurrences, GenIE's performance is close to 0. The model's performance slowly starts to rise for relations with at least $2^5 = 32$ occurrences, reaching a maximum of around 58% F1 for the few most frequent relations. Overall, the performance is higher than 50% for buckets that cover only 1.5% of the relations in the dataset. In contrast, training on Wiki-cIE Code, which has a uniform quality and coverage of annotations across relations, makes SynthIE perform well across all buckets. This translates to an F1 performance corresponding to a straight line at the top of the plot, at around 93%.

5.5 Summary and Critical Discussion

Summary. In this chapter, we show that useful data can be synthetically generated even for tasks that cannot be solved directly by LLMs: for problems with structured outputs, it is possible to prompt an LLM to perform the task in the reverse direction, by generating plausible input text for a target output structure. Leveraging this asymmetry in task difficulty makes it possible to produce large-scale, high-quality data for complex tasks. We demonstrate the effectiveness of this approach on cIE, where collecting ground-truth data is challenging, and no satisfactory dataset exists to date. We synthetically generate a dataset of 1.8M data points,

establish its superior quality compared to existing datasets in a human evaluation, and use it to finetune small models that outperform the prior state of the art (with equal model size) by a margin of 57 absolute points in micro-F1 and 79 points in macro-F1.

Implications for cIE. The lack of a large, balanced, high-quality dataset has been a significant obstacle for cIE. The synthetic data generated by the methodology proposed in this work satisfies all of the desiderata in terms of size, coverage, and quality (see Sec. 5.3.4 and Sec. 5.3.5). As an alternative to REBEL, which greatly suffers from both false positives and negatives, Wiki-cIE Text enables a substantially more accurate evaluation.
Similarly, Wiki-cIE Code enables the training of models performing well across all relations: (i) in terms of macro performance, SynthIE's predictions are of higher quality than REBEL's *gold annotations*; (ii) on the highest quality test set (Wiki-cIE Text), SynthIE pushes the macro-F1 score of 14%, for GenIE, to 93% (see Table 5.3 and Fig. 5.3).
While SynthIE's performance on REBEL Clean is better than the original *gold annotations*, it is still lower than the performance on Wiki-cIE Text. Analyzing the errors committed by our models and looking at the REBEL data allowed us to uncover an interesting property of cIE that, to the best of our knowledge, has not been identified by prior work. Assume that a model was trained on exhaustively annotated data (the holy grail which is now within reach) and presented with text containing central information that cannot be linked to the KB, (i) either in theory (e.g., "He plays the guitar") or (ii) under the output constraints assumed by the model. This would place the model in an undesired space of the likelihood distribution where it attempts to express information it cannot express, and is therefore bound to make mistakes. Fortunately, mistakes of this type can be avoided by (i) generating training data covering such scenarios and/or (ii) expanding the expressivity of models by extending the catalogs or developing novel output formats beyond subject, relation, object triplets. The latter goes towards exhaustive IE, whereas the former relaxes this desideratum. Overall, we believe that SDG could bring us closer to practical cIE systems. However, future work needs to re-evaluate and decide on the definition of *exhaustive* for practically useful *cIE*.

SDG and connection to self-improvement. This chapter highlights the efficacy of leveraging asymmetries in difficulty for SDG by developing a specific pipeline for cIE. However, the idea can be readily applied to different pipelines for cIE, or any other IE or parsing task, such as entity linking, oIE, or abstract meaning representation parsing. In fact, any problem with an inverse formulation that can be addressed more effectively by the LLM can benefit from our approach.
One influential application of this principle is STaR [296]. STaR assumes a dataset of question-answer pairs and proposes an iterative loop in which the model is fine-tuned on question-rational-answer pairs, where the rationales are generated by the model *and* lead to correct answers. Critically, the method improves the model's reasoning ability by bootstrapping from the existing capabilities thanks to an asymmetry created by passing not only the question but also the answer when generating the reasoning traces. In essence, STaR is one specific collabo-

ration between differently prompted instances of the model that leads to a self-improvement loop. We discuss such self-improvement collaborations more generally in Chapter 7.

Developing Collaborations Part IV

Preface

The preceding chapters focused on methods ensuring that LLMs can be effectively integrated into two-component collaborations. This ability opens up unprecedented opportunities for complex structured reasoning and collaboration among multiple AI systems, tools, and humans. To fully realize this potential, it is essential to develop a principled approach to designing, implementing, and studying such structured interactions. Chapter 6 introduces a conceptual framework and accompanying library that address this challenge.

6 Flows: Building Blocks of Reasoning and Collaborating AI

6.1 Introduction

The success of large language models (LLMs) largely lies in their remarkable emergent ability to adapt to information within their context (*i.e.*, prompt) [24, 268, 127]. By strategically crafting the context, LLMs can be conditioned to perform complex reasoning [268, 184] and effectively utilize external tools [220], significantly enhancing their capabilities. Some of the most exciting recent developments involve defining *control flows*, wherein LLMs, with the ability to control a set of tools, are called in an orchestrated fashion to solve increasingly complex tasks. Examples of such control flows include ReAct [286], AutoGPT [209], BabyAGI [177], PromptBreeder [67] and FunSearch [211]. Even the ubiquitous ChatGPT [185] application is an instance of a control flow built around the GPT-3.5 and GPT-4 models [24, 186]. However, these represent but a few of the many conceivable control flows, offering only a glimpse into the vast potential of structured LLM interactions. To realize this potential, we need to develop ways to study such interactions systematically.

In software engineering, simple processes can be implemented in an unstructured fashion, perhaps in a single file. However, as the size and complexity of the systems increase, choosing the right abstractions and architecture becomes critical [80]. Currently, for structured LLM interactions we want to model, implement, and study, we are at a point where this become unwieldy. Yet, no general efficient abstraction exists for effectively modeling arbitrarily complex structured interactions. Previous work and existing frameworks, such as LangChain [33], Chameleon [154], and HuggingGPT [229], have converged to an ad-hoc abstraction that models agents as entities that use LLMs to select and execute actions towards specific tasks, where the set of possible actions is pre-defined by the available tools. In this view, tools serve a narrow, well-defined goal and can perform sophisticated tasks (*e.g.*, querying a search engine or executing code). However, their behavior is limited to a single interaction. To highlight the implications of this limitation, consider the following scenario: Alice wants to apply for a job at HappyCorp. If Alice is an agent, she would need to explicitly plan the entire process, including preparing the application, sending it, and evaluating it, which may involve a back-

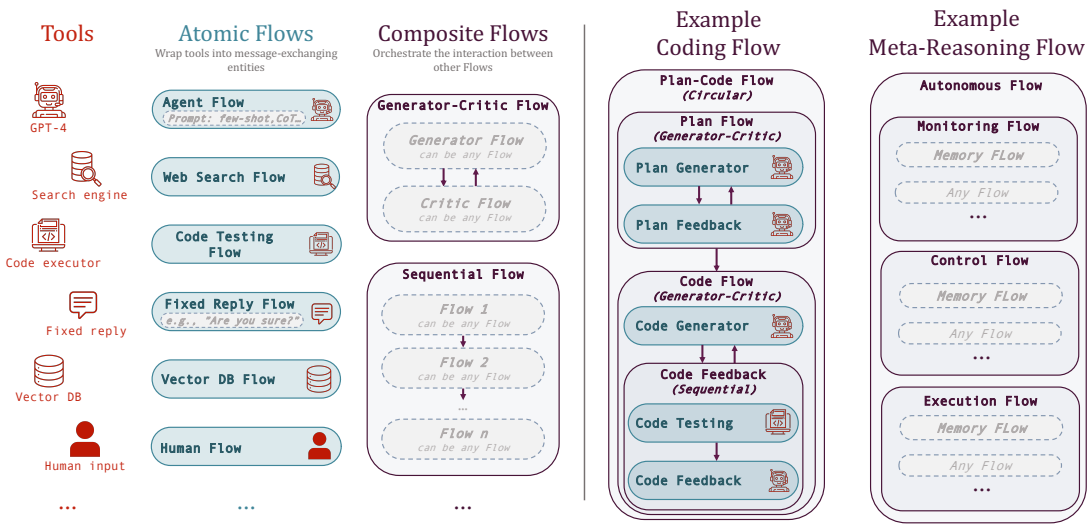


Figure 6.1: ***Flows* framework exemplified.** The first column depicts examples of tools. The second column depicts Atomic Flows constructed from the example tools. The third column depicts examples of Composite Flows defining structured interaction between Atomic or Composite Flows. The fourth column illustrates a specific Composite competitive coding Flow as those used in the experiments. The fifth column outlines the structure of a hypothetical Flow, defining a meta-reasoning process that could support autonomous behavior.[1]

ground check, organizing interviews, and more. Alice would need the knowledge and the "computational" ability to account for every detail, including unforeseen events that may arise (*e.g.*, the interviewer being on parental leave), and require her to adapt. In reality, most of the complexity is hidden from Alice behind an interface to HappyCorp's hiring process that might itself be composed of sub-processes involving many other *agents* and *tools*. Therefore, Alice is completely agnostic to the process(es) happening behind the interface and the respective logistics. On the other hand, the hiring process, carefully designed by experts, can be reused by many agents, and its sub-processes can be modified or improved with minimal or no impact on the other components beyond an updated interface. This makes it evident that agents and tools should be able to interact in complex, dynamic or static, ways as parts of nested, modular processes (that run locally or remotely), and the distinction between the two becomes blurred as they both serve as computational units in a complex computational process.

Starting from the observation that all processes are (control) flows defining a potentially complex interaction between many diverse components, we introduce a conceptual framework where Flows are the fundamental building blocks of computation. Flows are independent, self-contained, goal-driven entities able to complete semantically meaningful units of work. To exchange information, Flows communicate via a standardized message-based interface. The framework is depicted in Fig. 6.1.

The *Flows* abstraction ensures modularity. Alice, a higher-level meta-reasoning Flow that can support autonomous behavior, does not need to know anything beyond how to interface with

HappyCorp's hiring Flow. This substantially reduces complexity (Alice is interacting with a deeply nested, compositional structured interaction through a simple interface) and provides flexibility, allowing sub-Flows to be swapped without consequences as long as they have the same interface. Indeed, HappyCorp's pre-filtering Flow can be swapped from a rule-based system to an AI model or even a human Flow without affecting the structure of the overall process. The abstraction enables reusability and the composition of sub-Flows into new Flows for different tasks. Furthermore, the framework shares key design choices with the Actor model, one of the most prominent models of concurrent computation (*cf.* Sec. 6.3). Certainly, once Alice submits her application to HappyCorp, she does not need to wait for the response; she can move to her next goal while the other Flows run concurrently.

We showcase the potential of the proposed framework and library by investigating complex collaborative and structured reasoning patterns on the challenging task of competitive coding, a mind sport involving participants trying to solve problems defined by a natural language description.

Contributions. (i) We propose *Flows*, a conceptual framework providing an abstraction that simplifies the design and implementation of arbitrarily nested interactions while enabling concurrency. *Flows* can represent *any* interaction and provides a common framework for reasoning about interaction patterns, specifying hypotheses, and structuring research more broadly. (ii) We open-source the `aiFlows` library, which embodies *Flows*, together with FlowVerse, which is a repository of Flows that can be readily used, extended, and composed into novel, more complex Flows. (iii) We leverage *Flows* and the accompanying library to systematically investigate the benefits of complex interactions for solving competitive coding problems and develop AI-only Flows adding +21 and human–AI Flows adding +54 absolute points in terms of solve rate.

6.2 Related Work

Existing libraries for modeling structured interactions. LangChain [33] has become the go-to library for creating applications using LLMs. However, recent works involving structured interactions, such as Cameleon [154], Camel [142], HuggingGPT [229], and the concurrent works MetaGPT [103] and AutoGen [278] all come with their own library. Researchers opt to implement bespoke solutions due to the lack of a general yet efficient abstraction for modeling and designing structured interactions and the infrastructure to implement them that enable and facilitate open-ended exploration of novel ideas. In this work, we develop such an abstraction, *Flows*, which, in concert with `aiFlows`, fills this lacuna.

Impact of Flows. Crucially, the framework can implement any algorithm and efficiently covers all prior works on AI-AI, human-AI interactions, as well as prompt engineering (*cf.*

[1] For more details on meta-reasoning Flows see Sec. 6.7

Appendix D.3). These works focusing on specific Flow instantiations have demonstrated that structured interactions *can* yield performance gains across tasks and models. However, recent results put the universality of previously published results into question (e.g., Huang et al. [110]) and highlight the necessity for more systematic research. To support these research efforts, we develop the theoretical and practical infrastructure for modeling, implementation, and systematic study of structured interactions of arbitrary complexity. We demonstrate the benefits of the proposed infrastructure by conducting experiments that thoroughly investigate multiple core interaction patterns, including Human-AI collaboration, and their combinations, while accounting for data contamination and variance in the results, both of which are, surprisingly, not currently a standard.

Competitive coding (CC). One of the most prominent works addressing CC finetuned an LLM on GitHub code repositories and a dataset scraped from Codeforces [147]. Recently, Zelikman et al. [295] proposed decomposing CC problems into function descriptions and, for each function description, using an LLM to generate the implementation in a modular way. While these methods yield promising results, CC remains a challenging task far from being solved [186]. As such, it is an ideal test bed for studying the benefits of collaborative and reasoning interactions.

6.3 Flows

This section introduces *Flows* as a conceptual framework, describes its benefits, and presents the `aiFlows` library, which embodies the framework.

6.3.1 *Flows* as a Conceptual Framework

The framework is centered around *Flows* and *messages*. Flows represent the fundamental building block of computation. They are independent, self-contained, goal-driven entities able to complete a semantically meaningful unit of work. To exchange information, Flows communicate via a standardized message-based interface. Messages can be of any type the recipient Flow can process.

We differentiate between two types of Flows: Atomic and Composite.[2] Atomic Flows complete the work directly by leveraging *tools*. Tools can be as simple as a textual sequence specifying a (simple) Flow's fixed response or as complex as a compiler, a search engine, powerful AI systems like LLaMA [253, 252], Stable Diffusion [210], and GPT-4; or even a human. Notably, in the *Flows* framework, AI systems correspond to tools. An Atomic Flow is effectively a minimal wrapper around a tool and achieves two things: (i) it fully specifies the tool (e.g., the most basic Atomic Flow around GPT-4 would specify the prompts and the generation parameters); and (ii) it abstracts the complexity of the internal computation by exposing only a standard

[2]The concept of a Flow is sufficient for modeling any interaction. We introduce this distinction as it improves the exposition and simplifies the implementation.

message-based interface for exchanging information with other Flows. Examples of Atomic Flows include wrappers around chain-of-thought prompted GPT-4 for solving math reasoning problems, few-shot prompted LLaMA for question answering, a chatbot, a search engine API, or an interface with a human.

Composite Flows accomplish more challenging, higher-level goals by leveraging and coordinating other Flows. Crucially, thanks to their local state and standardized interface, Composite Flows can readily invoke Atomic Flows or other Composite Flows as part of compositional, structured interactions of arbitrary complexity. Enabling research on effective patterns of interaction is one of the main goals of our work. General examples of such patterns include (i) factorizing the problem into simpler problems (*i.e.*, divide and conquer); (ii) evaluating (sub-)solutions at inference time (*i.e.*, feedback); and (iii) incorporating external information or a tool. Importantly, Flows can readily invoke other, potentially heavily optimized, specialized Flows to complete specific (sub-)tasks as part of an interaction, leading to complicated behavior. One example of a Composite Flow is ReAct [286]. ReAct is a sequential Flow that structures the problem-solving procedure in two steps: a Flow selects the next action out of a predefined set of actions, and another Flow executes it. The two steps are performed until an answer is obtained. Another prominent example, AutoGPT, extends the ReAct Flow with a Memory Flow and an optional Human Feedback Flow. More generally, our framework provides a unified view of prior work, which we make explicit in Appendix D.3.

Importantly, as illustrated in Fig. 6.1, Composite Flows can script an arbitrarily complex pattern (i) precisely specifying an interaction (*e.g.*, generate code, execute tests, brainstorm potential reasons for failure, etc.); or (ii) defining a high-level, meta-reasoning process in which a Flow could bring about dynamic unconstrained interactions.

Key properties. The proposed framework is characterized by the following key properties: 1) Flows are the compositional building blocks of computation; 2) Flows encapsulate a local, isolated state; 3) Flows interact only via messages; 4) Flows' behavior depends only on their internal state and the input message; 5) Flows can send messages to other Flows and create new Flows.

Connection to the *Actor* model. *Flows* is fundamentally a framework modeling the computation underlying interactions. As such, it shares key design principles with the *Actor* model [99]—a mathematical model of concurrent computation. Similarly to *Flows*, in the *Actor* model, an Actor is a concurrent computation entity that can communicate with other Actors exclusively through an asynchronous message-passing interface. By encapsulating the state and the computation within individual Actors, the model provides a high-level abstraction for effectively managing and reasoning about complex concurrent and distributed systems, completely avoiding issues associated with shared states, race conditions, and deadlocks. These benefits are similar in nature to those observed in the domain of interactions. The main distinction between the proposed framework and the *Actor* model lies in their respective communication protocols. Concretely, while the *Actor* model prescribes purely asynchronous

communication, *Flows* natively supports synchronous communication, which is essential for the implementation of structured reasoning. Interestingly, a similar deviation from the "pure" *Actor* model can be identified in the implementation of Erlang, a concurrent programming language based on it [9]. Overall, the shared design choices still make *Flows* inherently concurrency-friendly, from the practical perspective, and suffice for important results from the five decades of extensive studies of the *Actor* model, such as the fact that every physically possible computation can be directly implemented using Actors [98], to transfer to *Flows*.

6.3.2 Why *Flows*?

Modularity. *Flows* introduces a higher-level abstraction that isolates the state of individual Flows and specifies message-based communication as the only interface through which Flows can interact. This ensures perfect modularity by design.

Reduction of complexity. The framework ensures the complexity of the computation performed by a Flow is fully abstracted behind the universal message-based interface. This enables an intuitive and simple design of arbitrarily complex interactions from basic building blocks.

Systematicity, flexibility, and reusability. The separation of responsibility allows modules to be developed and studied systematically in isolation or as part of different interactions. Once the correctness and the benefits of a Flow are established, it can be readily used in developing novel Flows or as a drop-in replacement for less effective Flows leveraged in completing similar goals.

Concurrency. The proposed framework's design is consistent with the Actor model, one of the most prominent models of concurrent computation. As a consequence, *Flows* can readily support any setting in which Flows run concurrently.

6.3.3 The `aiFlows` Library

Accompanying *Flows*, we release the `aiFlows` library, which embodies the framework. In addition to the inherent benefits that come with the framework, the library comes with the following add-ons: (i) FlowVerse: a repository (to which anyone can contribute) of Flows that can be readily used, extended, or composed into novel, more complex Flows. *Flows* allows for existing "tools" (as well as "models", "chains", "agents", *etc.*) to be readily incorporated by wrapping them in an Atomic Flow; (ii) a detailed logging infrastructure enabling transparent debugging, analysis, and research in optimizing (i.e., learning or fine-tuning) Flows.

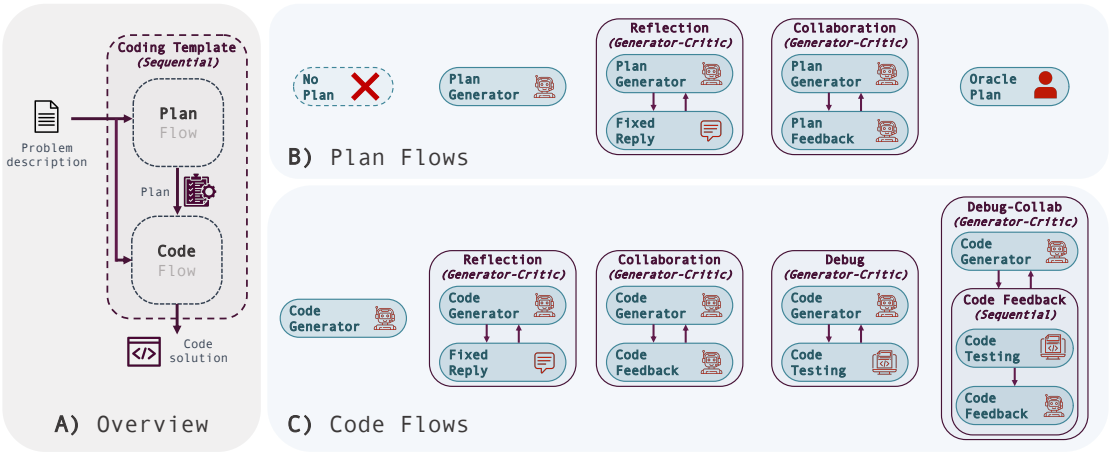


Figure 6.2: **Competitive coding Flows.** At the highest level, we consider planning as a specific structured reasoning pattern for problem decomposition. In particular, the Plan Flow generates a solution strategy and passes it to the Code Flow, which implements it, as depicted in A). B) and C) depict the different choices of sub-Flows used as Plan and Code Flows in the experiments. Notably, we explore the impact of human-AI collaboration at the plan level and refinement with different types of *feedback*: i) fixed reply encouraging reflection; ii) AI generated feedback; iii) code testing results as feedback; iv) AI generated feedback grounded in code testing results.

6.4 Competitive Coding Flows

This work investigates the potential of structured interactions for solving competitive coding (CC) problems. In CC, given a natural language description and a few input–output examples, the task is to generate code that will produce the expected output for all of the hidden input–output test cases associated with the problem. Fig. D.1 provides examples.

We focus the analysis on three canonical dimensions of interactions: (i) problem decomposition as structured reasoning; (ii) human-AI collaboration; and (iii) refinement with various feedback types. By providing a common language for clearly specifying interactions as well as the capability to flexibly compose, exchange, and extend them, the framework makes it possible to study the space of complex interactions in a principled fashion. In the rest of the section, we describe the specific Flows used in the experiments, depicted in Fig. 6.2.

Problem decomposition. Planning has been an integral intermediate step in recent work [154, 229, 286]. Similar decomposition is natural in the context of CC as well. In particular, we approach the task in two steps: generating a solution strategy by a Plan Flow and then generating the corresponding code by a Code Flow. This is depicted by panel A in Fig. 6.2.

Human-AI collaboration. When designing human-AI collaborations, it is essential to take the costs of human interaction into account [105, 3, 176]. By providing immense flexibility, *Flows* can support research in the design of interactions involving humans as computational building

blocks in a way that maximizes the utility of the overall computation with a minimal human effort. In the context of CC, we hypothesize that a human can be effectively incorporated at the plan level to provide a short "oracle" plan in natural language. We operationalize this by an (Atomic) Human Flow, illustrated in Panel B of Fig. 6.2 as the *Oracle Plan* Flow.

Refinement with various feedback types. Iterative refinement is a general problem-solving strategy successfully deployed across various disciplines [192, 205, 219, 215]. The strategy revolves around the idea that a solution can be gradually improved through a mechanism for analysis, modification, and re-evaluation. The design of this "*feedback*" mechanism is critical for the effectiveness of the problem-solving strategy. The conceptual framework, paired with the accompanying library, provides the infrastructure to support the design, implementation, and principled research of effective refinement strategies and feedback mechanisms. In this work, we consider a canonical iterative refinement setup where a *generator* Flow is tasked with generating the solution, and a *critic* Flow provides feedback on the proposed solution. We consider two feedback types in the context of both the Plan and the Code Flow: (i) Reflection Flow: the feedback consists of a fixed message encouraging the model to reflect on important aspects of the proposed solution; (ii) Collaboration Flow: the feedback is provided by an AI system that "evaluates" the proposed solution. Furthermore, we explore two more code-specific feedback types: (i) Debug Flow: the feedback message corresponds to the results from executing the code and testing it against the examples provided in the problem description; (ii) Debug–Collab Flow: the feedback is provided by an AI system with access to the code testing results, effectively, grounding the feedback and allowing more systematic reasoning about the potential causes of failure.

We refer to Flows using the following convention: *CodeFlowName* when no plan is generated and *PlanFlowName-CodeFlowName* otherwise.

6.5 Experimental Setup

Data. We scrape publicly available problems from one of the most popular websites hosting CC contests, Codeforces [174], and LeetCode [139], which cover a broad spectrum of problems ranging from easy interview questions to hard CC problems (see Appendix D.1 for more details). The datasets cover problems from 2020-August-21 to 2023-March-26 for CodeForces, and from 2013-October-25 to 2023-April-09 for LeetCode. Importantly, to study the effect of structured interactions (*i.e.*, different Flows) in a principled manner, it is crucial to account for the possibility of *data contamination*, *i.e.*, that some of the test data has been seen during training [158]. Containing problems published over an extended period up to a few months ago (at the time of writing), our datasets allow for reliable identification of the training data cutoff date that can help with addressing this issue. Prior code evaluation datasets like APPS [97], HumanEval [37], and CodeContests [147] lack problem release dates, and considering the lack of publicly available information about LLMs' training data, can likely lead to confounded evaluation of models' memorization and generalization abilities.

Code testing and solution evaluation. Just like a human participant, the Debug Flow has access only to the input–output example pairs contained in the problem description and, at inference time, uses a local code testing infrastructure to evaluate (intermediate) solution candidates. Crucially, these examples cover only a few simple cases, and generating outputs consistent with them does not imply the code corresponds to a correct solution. A solution is considered correct if it passes all the hidden test cases. To determine correctness, we leverage online evaluators that submit candidate solutions to the websites' online judges, ensuring authoritative results. For many of the Codeforces problems, we also support local evaluation based on a comprehensive set of hidden test cases we managed to scrape. For more details, see Appendix D.2.

Models and Flows. We experiment with the competitive coding Flows described in Sec. 6.4, and GPT-4 [186] as the LLM tool of choice. See Appendix D.4 for the specific prompts. Also, the code to reproduce our experiments is available in the project's GitHub repository.

Evaluation metrics. The most common evaluation metric for code generation is pass@k, corresponding to the probability that in a set of k sampled candidates, there will be at least one correct solution [37]. To better align with practical use cases, we focus on pass@1, *i.e.* the solve rate when averaged across the problem set. We report a point estimate and a 95% confidence interval constructed from 1000 bootstrap resamples.

Compute and cost. All the experiments, including the most complex Flows, can be performed on commodity hardware relatively cheaply. For instance, the costs associated with querying the OpenAI API for generating Table 6.1 amount to $1000.

6.6 Experimental Results

We first study the generalization of representative Flows and empirically identify GPT-4's knowledge-cutoff date. Next, we perform a focused analysis along the dimensions described in Sec. 6.4.

6.6.1 Performance of Coding Flows on Pre- vs. Post-Knowledge-Cutoff-Date Data

In this experiment, we consider three representative Flows: (i) Code: the simplest Code Generator Flow corresponding to a single GPT-4 API call; (ii) Code_Debug_Collab: the most complex code Flow; (iii) Plan_Oracle-Code_Debug_Collab: the most complex code Flow with human guidance at the plan level. We perform the analysis by running the three Flows on Codeforces problems released from October 2020 to April 2023 and averaging the performance over a sliding window of two months. The results are reported in Fig. 6.3.

We observe a substantial drop in performance centered around September 2021, consistent with the knowledge cutoff date reported by OpenAI, and denote it by a vertical line on the plot. With Codeforces problems appearing in contexts outside of the contest itself (*e.g.*, editorials),

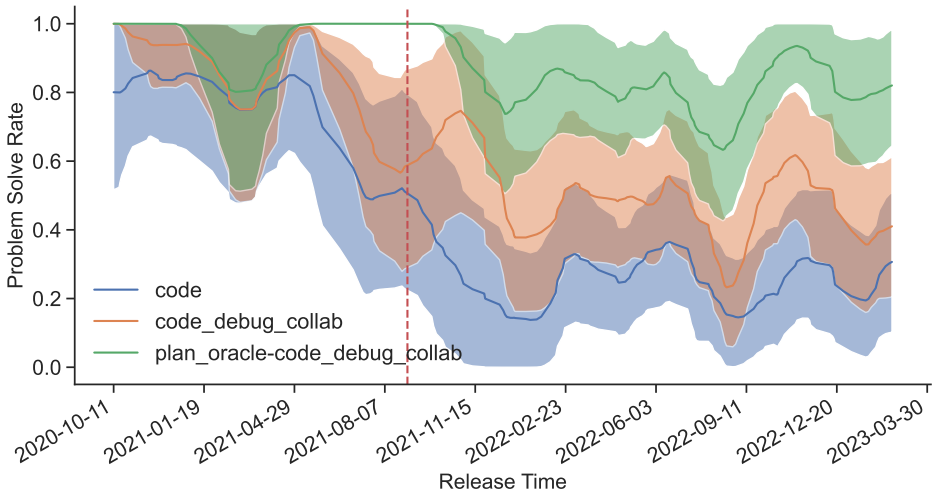


Figure 6.3: **Temporal analysis.** Performance is averaged over a sliding window of two months. The substantial drop in performance around the reported knowledge cutoff date for GPT-3/4 (the crimson line) reveals limited generalization ability that can be alleviated through structured interactions.

it is reasonable to assume the model has been exposed to older problems more frequently during training. This would explain why the drop spans multiple months, from May 2021 to November 2021, depending on when which data was published and crawled.

Notably, there is a stark difference in the performance of the Code Flow on problems published before and after the knowledge cutoff data, with the solve rate decreasing from around 80% to 23%. While still experiencing a substantial performance drop, the Code_Debug_Collab Flow doubles the solve rate on novel problems to around 45%. Provided with human input at the plan level, the same Flow reaches 85%. Overall, this highlights that GPT-4 performs poorly on novel complex reasoning problems, but structured interactions have the potential to enhance its generalization capabilities. As both GPT-4 (*i.e.*, the Code Flow) and the more complex interactions (Flows) exhibit qualitatively different behavior on novel data, to draw accurate conclusions, it is critical that data contamination is taken into serious consideration when designing experiments and interpreting results.

6.6.2 Comparing Competitive Coding Flows

Table 6.1 reports the performance of the systematically chosen set of Flows described in Sec. 6.4. Rows 6–10 correspond to Flows comprising planning and coding, while rows 1–5 perform the coding directly. In line with the findings of the previous section, we separately consider the performance on problems published before and after the knowledge cutoff date of September 2021.

	Codeforces		Leetcode					
	Pre-cutoff	Post-cutoff	Pre-cutoff			Post-cutoff		
			Easy	Medium	Hard	Easy	Medium	Hard
Code	71.8 ±11.0	26.9 ±11.0	97.8 ±3.1	93.4 ±5.4	66.7 ±10.9	76.3 ±8.6	25.1 ±8.9	8.0 ±5.5
Code_Reflection	+9.3 ±9.7	+0.0 ±10.6	+0.0 ±3.1	+0.0 ±5.4	+1.2 ±10.6	+0.9 ±8.1	+5.4 ±9.4	+3.5 ±6.6
Code_Collaboration	+4.8 ±10.5	+9.6 ±11.8	+0.0 ±3.1	-2.3 ±6.0	-0.1 ±10.9	-3.2 ±8.7	+0.0 ±8.7	+1.2 ±5.9
Code_Debug	+12.7 ±8.6	+7.9 ±11.6	+0.0 ±3.1	+1.1 ±5.0	+6.9 ±10.0	+7.7 ±7.3	+7.7 ±9.6	+2.4 ±6.3
Code_Debug_Collab	+12.6 ±8.9	+20.6 ±12.1	+0.0 ±3.1	+0.0 ±5.4	+5.5 ±10.4	+7.5 ±7.4	+9.8 ±9.7	+1.2 ±6.0
Plan-Code	-1.6 ±11.0	+8.0 ±11.6	-3.1 ±4.5	-2.3 ±5.9	-9.7 ±11.2	+2.3 ±8.3	+3.2 ±9.1	-3.4 ±4.3
Plan_Reflection-Code	-3.3 ±11.6	+4.8 ±11.6	-2.1 ±4.1	-4.5 ±6.6	-3.1 ±10.7	+1.2 ±8.3	-3.3 ±8.5	+0.0 ±5.5
Plan_Collaboration-Code	-4.8 ±11.5	+6.3 ±11.4	-1.1 ±3.7	-2.3 ±6.1	-7.2 ±11.2	-2.0 ±8.6	+0.1 ±9.0	+1.2 ±5.8
Plan_Oracle-Code	+11.0 ±9.4	+47.6 ±10.7	–	–	–	–	–	–
Plan_Oracle-Code_ Debug_Collab	+23.0 ±5.2	+53.9 ±9.5	–	–	–	–	–	–

Table 6.1: **Main Results.** Performance of competitive coding Flows on Codeforces and LeetCode, with direct inference (Code) as baseline.

Problem decomposition. The idea behind planning before implementing the solution is to decouple the high-level reasoning from the code implementation. To analyze the effectiveness of this pattern, we compare the Code and the Plan-Code Flow. Looking at the point estimates, in the pre-cutoff problems, introducing the plan Flow leads to decreased performance (-1.6 for Codeforces and -3.1/2.3/-9.7 for LeetCode easy/medium/hard). However, in the post-cutoff problems, incorporating a plan Flow leads to gains for Codeforces (+8) and LeetCode easy and medium (+2.3 and +3.2). While these trends are consistent, considering the confidence intervals, we see that they are not statistically significant. Crucially, these results do not imply that this specific problem decomposition is not valuable as it creates a lot of potential in designing an effective human-AI collaboration.

Human-AI collaboration. After every contest, the Codeforces community publishes an editorial that, in addition to the code implementation, provides a short natural language description of the solution. To simulate a Flow where a human provides high-level guidance at the core of the reasoning process, we scrape the solution descriptions and pass them as human-generated plans. The results are striking: despite being only a few sentences long, human-provided plans lead to a substantial performance increase (from 26.9% to 74.5% and from 47.5% to 80.8% on novel problems, when the code is generated by Code and Code_Debug_Collab Flows, respectively). First and foremost, these results showcase the opportunities created by *Flows* for designing, implementing, and studying Human-AI collaboration as a key component of structured interactions. Second, specific to the problem of competitive coding, they validate the hypothesis that high-quality plans are important, suggesting that the design of more effective plan Flows is a promising direction to explore in the future. Last but not least, the results highlight the necessity of more systematic research, as patterns seemingly not valuable in one Flow, such as the simple plan-code structured reasoning problem decomposition, can provide immense value as part of another Flow.

Refinement with various feedback types. We find that Code_Reflection and Code_Collaboration lead to limited improvements among the code Flows. The two exceptions are Codeforces pre-cutoff (+9.3) for the former and Codeforces post-cutoff (+9.6) for the latter pattern. While close, these results are not statistically significant. On the other hand, the Flows providing grounded feedback, Code_Debug and Code_Debug_Collab, lead to consistent and statistically significant improvements, most notable on the novel Codeforces problems where performance increases from 26.9, without feedback, to 47.5, when the refinement is based on AI-generated feedback grounded in tests. On LeetCode, these improvements are smaller in magnitude. We suspect this is a consequence of the examples provided with the problem description being more simple than those in Codeforces, leading to false positives and, thereby, incorrect grounding, affecting the feedback quality. This could be addressed by generating additional tests with a Test_Case_Generator Flow, a direction we leave for future work to explore. Finally, in the plan Flows, where we consider Reflection and Collaboration (without grounding), we find that refinement does not provide statistically significant benefits.

Overall, our findings provide several important insights: (i) the direct benefit of problem decomposition hinges on the quality of the intermediate steps; (ii) involving humans at the core high-level reasoning process yields major improvements as humans can easily provide high-quality, grounded feedback; (iii) strategic problem decomposition is a powerful strategy for creating opportunities for effective Human–AI collaboration; (iv) the effectiveness of refinement patterns is not universal and depends on the quality of the starting solution and the feedback (*e.g.*, the level of grounding), and the model's ability to incorporate that feedback modulated through the feedback's specificity and the model's capabilities. This analysis paints a more complex picture than what is reported by prior work.

6.7 Summary and Critical Discussion

Summary. We propose *Flows*, an abstraction that, in concert with the accompanying library `aiFlows`, provides a theoretical and practical infrastructure with modular and concurrency-friendly design, which enables the modeling, implementation, and systematic study of arbitrarily complex structured interactions. We thoroughly investigate multiple core interaction patterns, including Human-AI collaboration, and their combinations, on the task of competitive coding while accounting for data contamination and the variance in the results. The investigation shows that the developed AI-only Flows add +21 and human–AI Flows add +54 absolute points in terms of solve rate and highlights the effect of data contamination, variance, and non-universality of results. Overall, we empirically establish the potential of structured interactions, *i.e.*., Flows, the necessity of more systematic research, and the value brought by *Flows* and `aiFlows` to support these research efforts.

Implications. To the best of our knowledge, this work is the first to address the challenges associated with the *development and execution* of complex structured interactions. Crucially,

our goal went beyond streamlining the process of implementing simple interactions, which was the focus of the community at the time of writing, and centered on an abstraction necessary to *enable and support* the development and execution of *arbitrary* interactions. For example, distributed and asynchronous execution of Flows such as FunSearch [211] is readily supported by `aiFlows`. This infrastructure naturally creates research opportunities in many areas, such as developing Flows for synthetic data generation or autonomous AI agents, which we discuss in the next chapter.

On the other hand, thanks to its key properties, *Flows*, together with `aiFlows`, greatly simplifies the design and implementation of open-ended interactions, with a capability to flexibly isolate, compose, replace, or modify sub-Flows, facilitating research. We demonstrate the utility of the infrastructure by performing a thorough evaluation using a single model and a specific subset of interactions on the task of competitive coding. Our experiments show that carefully designed interactions can substantially improve generalization. However, they also reveal that the effectiveness of particular interaction patterns is not universal; instead, many factors are at play, highlighting the necessity for more research. As researchers, we need to clearly specify the patterns we are studying, clearly communicate our hypotheses, and study them both in isolation and as parts of other interactions across different datasets and tasks. Furthermore, it is critical that data contamination is taken into serious consideration when designing experiments and drawing conclusions, and error bars become a standard in the field.
We hope the framework will serve as a solid basis for practical and theoretical innovations, paving the way toward ever more useful AI, similar to the Actor model's role for concurrent and distributed systems.

Part V
Outlook and Conclusion

Preface

Parts II, III, and IV establish the infrastructure for developing arbitrarily complex structured interactions between AI systems, tools, and humans. Building on this foundation, this part explores how to leverage this infrastructure to build increasingly powerful AI systems. Specifically, we examine how these structured interactions can be optimized, moving beyond enabling such interactions to understanding how to maximize their effectiveness.

Chapter 7 proposes the semantic decoding perspective that allows us to systematically study the vast design space of semantic decoding algorithms based on the optimization they perform. Specifically, this perspective conceptualizes LLMs as one among a large pool of semantic processors, including humans and tools, such as search engines or code executors. Collectively, semantic processors engage in dynamic exchanges of semantic tokens to progressively construct high-utility outputs. We explore the possibilities of optimizing within the space of semantic tokens via semantic decoding algorithms. By focusing on the semantic level and disregarding syntactic details, we gain a fresh perspective on the engineering of AI systems, allowing us to imagine systems with much greater complexity and capabilities. We conclude this chapter with a list of research opportunities and questions emerging from the work presented in this thesis.

In Chapter 8, we provide a conclusion to the thesis.

7 Flows as Semantic Decoding Algorithms

7.1 Introduction

Recent research suggests that strategically orchestrated collaborations between large language models (LLMs), tools, and humans can effectively overcome LLMs' inherent limitations, leading to substantial performance improvements [223, 212, 59, 287, 17, 262, 264, 232, 50, 61].

To conceptualize this evolution, one can consider LLMs as generators of semantically coherent units, often referred to as *thoughts* or, equivalently in this work, *semantic tokens* [267, 287, 17, 59, 223]. This viewpoint positions LLMs as just another kind of contributor to a diverse pool of what we call *semantic processors*, which includes humans, search engines, external memories, code executors, and more. Collectively, these semantic processors engage in a dynamic process, exchanging and manipulating semantic tokens to progressively construct a high-utility semantic token as output [59]. To structure the vast space of possible collaboration strategies, several frameworks have been proposed, such as aiFlows (see Chapter 6), LangChain [33], [118], MetaGPT [103], SwarmGPT [115], and AutoGen [278], among others.

This work presents a different perspective on the advancements in AI collaboration, independent of, but consistent with, these frameworks. Rather than proposing an abstract model of communication between semantic processors, we focus on the optimization that the interaction is globally performing in semantic space to search for the solution. We call this perspective *semantic decoding* because it views semantic tokens as the basic units of a new type of computation happening directly in the space of semantic tokens. Then, a *semantic decoding algorithm* is an orchestrated interaction between semantic processors that performs optimization and search in semantic space to reliably construct high-utility trajectories. Our perspective is visually summarized in Fig. 7.1.

This optimization perspective draws a direct analogy with the well-known problem of *syntactic decoding*, where an algorithmic process—the decoding algorithm—aims to extract a high-utility sequence of (syntactic) tokens while being guided by the next token distribution of an auto-regressive language model. To build around the inherent limitations of auto-regressive

language models, as discussed in Chapter 4, the field of syntactic decoding produced a rich set of techniques leveraging external information and heuristics to guide the search in the space of token sequences, optimizing for utility [160, 163, 120, 129, 95, 130, 29].

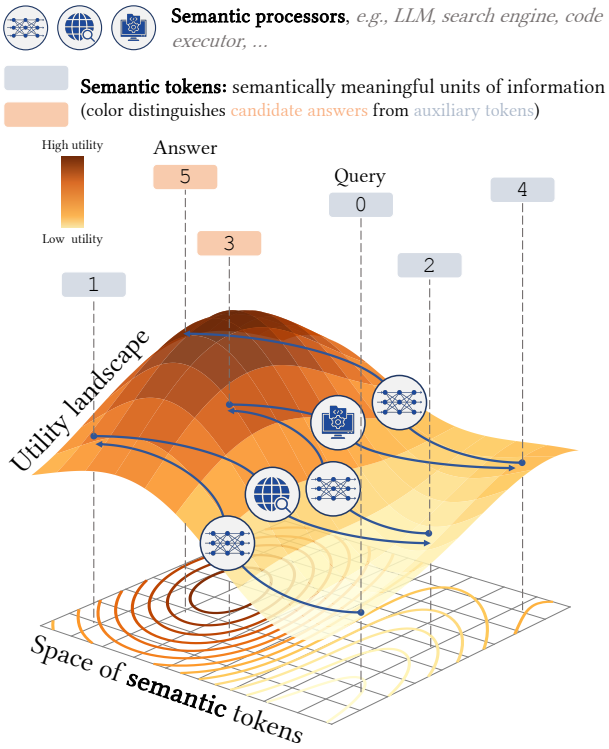


Figure 7.1: **Illustration of semantic decoding: optimizing utility in the space of semantic tokens.** Semantic tokens— semantically coherent units of text— form the basic units of communication among what we call **semantic processors**, which includes LLMs, humans, and various tools. Then, **utility** is a function defined over the space of semantic tokens, indicating a semantic token (or stream of tokens) solves the task. A **semantic decoding algorithm** orchestrates the exchange of semantic tokens among semantic processors to robustly extract a high-utility semantic token. This orchestration can be viewed as a search and optimization procedure within the semantic space. Throughout the decoding process, auxiliary tokens (depicted in gray) are generated; while these tokens are not answers themselves and have low utility, they serve as anchor points for further exploration toward regions of higher utility. Examples of auxiliary tokens include feedback or grounding information. The generation of auxiliary tokens should increase the expected utility of the trajectory in the semantic space. This example is a simplified illustration of a basic trajectory in the semantic space. We show in Sec. 7.4 and Sec. 7.5 that semantic decoding can be used in much more complex and creative ways.

The shift from syntactic to semantic decoding may, at first glance, appear trivial because, clearly, a semantic token is just a sequence of syntactic tokens that language models were designed to produce anyway. However, by abstracting away from the syntactic details and focusing on the computation in the semantic space, a fresh perspective on the engineering

of AI systems and their capabilities emerges. This opens the door to imagining systems with much greater complexity. To highlight the importance of good abstractions for innovation, note that the development of modern LLMs would have been infeasible if one had to program them directly in terms of sequences of bits. Not only would this task be overwhelmingly complex for humans to do, but without the proper mental abstractions, it would have been impossible to even imagine something as complex as an LLM. In our view, the field is now ready to move to yet another powerful abstraction where semantic, rather than syntactic, tokens become the building blocks of a new type of computation, one that operates directly in the space of meaningful thoughts and concepts.

In this article, we begin by articulating the transition from manipulating syntactic tokens to manipulating semantic tokens (Sec. 7.2). Notably, we underscore that the combination of a language model with a syntactic decoding algorithm creates an engine capable of interpreting semantic tokens as input and generating semantic tokens as output, effectively transforming it into a semantic processor.

Moving on to Sec. 7.3, we draw the analogy between syntactic and semantic decoding. Specifically, we introduce a generalized notion of a decoding algorithm as an algorithmic layer that operates on top of token processors. This layer orchestrates search and optimization over tokens to robustly extract high-utility outputs. We argue that semantic decoding is a pragmatic computation because the optimization of utility via the exchange of semantic tokens is a computation that gives rise to a dynamic and task-dependent notion of meaning.

In Sec. 7.4, our focus shifts to semantic decoding algorithms, where we categorize the types of optimization performed in semantic space into three distinct groups:

- **Grammars of thoughts**, encompassing fixed heuristic patterns such as chain-of-thoughts [267], planning before implementing [263, 264, 78, 190, 118], or relying on fixed feedback or grounding mechanisms [270, 190, 118]. These approaches can be loosely perceived as the semantic-level generalization of *grammar-constraint decoding* at the syntactic level [255, 221, 213, 85].

- **Guided search**, representing methods that sample and search the semantic space while being guided by *value models*. Noteworthy examples include Tree-of-Thought [287] and FunSearch [212], both utilizing large language models (LLMs) to sample semantic tokens and guiding the overall decoding process with value models. This category is the semantic equivalent of *value-guided beam search* (VGBS) [95, 206, 130] and *Monte Carlo tree search* (MCTS) variants [29, 120].

- **Learning to optimize**, comprising methods that fully embrace the optimization perspective by learning effective ways to navigate the semantic space. Currently underexplored, this category covers methods in which individual components are trained or fine-tuned to be better collaborators, or controllers to route semantic tokens to appropriate seman-

tic processors at the correct time step. These represent the semantic-level counterparts of the paradigm of *learning to decode* at the syntactic level [273, 45].

In Sec. 7.5, we present an extensive, albeit non-exhaustive, list of research opportunities and questions emerging from the semantic decoding perspective, enabled by the work in the rest of the thesis. This encompasses topics such as (meta-)prompt engineering, learning to optimize in semantic space, synthetic data flows, human–computer collaboration, evaluation, interpretability, control, ethics of semantic decoding algorithms, as well as the infrastructure necessary to support such developments.

To help the reader, we provide a glossary of key terms in Sec. E.1. Additionally, each section concludes with a concise summary highlighting the key points for easy reference.

7.2 From Syntactic to Semantic Tokens

At its core, computation is a syntactic process where fundamental information building blocks are manipulated. These building blocks might take the form of binary digits (bits), abstract symbols in some computational models, or tokens in language models. As Claude Shannon himself noted, computation is syntactic because the symbols lack inherent *meaning* [225]; instead, meaning arises externally through context, via the processes that manipulate them, and the outcomes they produce for external actors.

Since Shannon, many have tried to lift the theory of information processing from the syntactic to the semantic level. The prevailing idea is to shift to computational models wherein basic symbols inherently carry semantic content, readily understandable, with meaningful impact on the actors outside the computation [16, 27, 194]. This idea is illustrated by the concept of *semantic units* in the semantic theory of information [300, 69], or the proposal of a *language of thoughts* [70], postulating that thinking operates on atomic units of meaningful content [207].

The shift discussed in this article is of a similar kind, moving from language models processing syntactic tokens to interactions between LLMs and tools whose basic units of computation are *semantic tokens*, i.e., concepts intelligible for users outside of the computation. These semantic tokens are not abstract symbols awaiting to be interpreted, but active carriers of meaning, embodying concepts and ideas directly. We now describe formally syntactic and semantic tokens.

7.2.1 Syntactic Tokens

Syntactic tokens serve as the fundamental computational building blocks in modern natural language processing systems. The finite collection of all possible tokens forms a *syntactic vocabulary* Σ. These syntactic tokens are designed to be assembled into sequences through concatenation. Let $\mathbf{x} \in \Sigma^*$ represent one such sequence, defined as $\mathbf{x} := \langle x_0, \ldots, x_m \rangle$. Typically,

syntactic tokens may consist of words or characters, but more commonly, they are subword units. The set of these units is often learned based on the frequency of character co-occurrences in the available training data with methods such as byte-pair encoding (BPE) [72, 224]. The algorithm that breaks down natural language into a set of tokens and therefore defines the vocabulary Σ is called the *tokenizer*.

7.2.2 Syntactic Token Processors: Language Models

Syntactic tokens are symbols, and language models are computational processes that manipulate them. In particular, a probabilistic language model (PLM) induces a probability distribution P over all possible strings, Σ^*, that can be constructed from the vocabulary of syntactic tokens Σ. The purpose of the language model is to read an input sequence $\mathbf{x}$ of tokens and guide the assembling of an output sequence $\mathbf{y}$.

To efficiently represent a probability distribution over the large combinatorial space of all possible strings, language models use an auto-regressive decomposition, meaning that they model the probability of each subsequent token given a sequence of previous tokens: $P(y_t|\mathbf{y}_{<t})$, where $\mathbf{y}_{<t} := \langle y_0, \ldots, y_{t-1} \rangle$. Most modern language models are parameterized by a Transformer architecture with trainable weights θ [257]. Then, the probability distribution of an output sequence $\mathbf{y} := \langle y_0, \ldots, y_n \rangle$, potentially conditioned on an input $\mathbf{x} := \langle x_0, \ldots, x_m \rangle$, is given by

$$P(\mathbf{y}|\mathbf{x}) = \prod_{t=0}^{|\mathbf{y}|} p_\theta(y_t|\mathbf{y}_{<t}, \mathbf{x}). \tag{7.1}$$

Importantly, the language model alone does not directly tell us how to produce an output sequence; it only specifies a probability distribution over the next token given a partial sequence. As described in Sec. 7.3, combining it with a decoding algorithm is necessary to transform a language model into a system that can produce output sequences.

7.2.3 Semantic Tokens

We define a *semantic token*, also referred to as a *thought*, as a sequence of syntactic tokens that conveys *meaningful information*. A semantic token, denoted as $\mathbf{x}^\sigma \in \Gamma$, is an element of *semantic vocabulary* Γ, a subset of Σ^*, representing the potentially infinite set of semantic tokens. It is important to note that not all syntactically valid strings convey meaningful information; thus, $\Gamma \neq \Sigma^*$. To differentiate semantic tokens (e.g., $\mathbf{x}^\sigma$) from arbitrary strings (e.g., $\mathbf{x}$), we use the superscript σ.

What exactly defines a semantic token? When can we determine that a string is "semantically meaningful"? Semantic tokens are embedded within natural language, drawing their meaning directly from potential human interpretation. Similar to how words derive meaning through contextual usage within a language, observed by external actors, the meaning of a string is shaped by its interaction with other elements in the system and its relationship with other

semantic tokens. Finally, they also carry *pragmatic meaning* due to the effects they induce on the computation. For instance, both the input and output sequences of an AI system are semantic tokens, with meaning assigned through their usage. The input sequence $\mathbf{x}^\sigma$ is *intended to* prompt a specific response or behavior from the AI system, while the output sequence $\mathbf{y}^\sigma$ is *interpreted as* a response to a query, serving a particular purpose. Semantic tokens extend beyond input and output sequences; they encompass any other text conveying meaningful information. This includes thoughts, as defined and utilized in various frameworks such as chain-of-thoughts [267], tree-of-thought [287, 153, 282], and graph-of-thoughts [17, 289].

7.2.4 Semantic Token Processors

Probabilistic semantic token model. A probabilistic language model is trivially also a *probabilistic semantic token model*. Due to the auto-regressive decomposition, the language model induces a probability over strings and, therefore, over thoughts: $p_\theta(\mathbf{y}^\sigma|\mathbf{x}^\sigma)$. In general, if there is a history of previously generated semantic tokens $H = [\mathbf{x}_0^\sigma, \ldots, \mathbf{x}_m^\sigma]$, then the language model induces a probability distribution on the next semantic token:

$$P(\mathbf{y}^\sigma|H) = P(\mathbf{y}^\sigma|\mathbf{x}_0^\sigma, \ldots, \mathbf{x}_m^\sigma), \tag{7.2}$$

which is, under the hood, still decomposed via an auto-regressive model on syntactic tokens:

$$P(\mathbf{y}^\sigma|H) = \prod_{t=0}^{|\mathbf{y}|} p_\theta(y_t|\mathbf{y}_{<t}, \mathbf{x}_0^\sigma, \ldots, \mathbf{x}_m^\sigma). \tag{7.3}$$

In fact, this property makes the perspective shift from syntactic to semantic token possible, the language model being the enabler of the shift.

From language models to semantic processors. A *semantic token processor*, or *semantic processor* in short, is a system designed to take a semantic token $\mathbf{x}^\sigma$ as input and generate a corresponding output semantic token $\mathbf{y}^\sigma$. While numerous systems fall into this category, let us focus on semantic processors based on a language model. The language model, by itself, provides only a distribution over the next syntactic tokens. However, to generate a sequence of syntactic tokens, it needs to be paired with a *decoding algorithm*. Essentially, a decoding algorithm is responsible for utilizing the next token distribution and determining which syntactic tokens to assemble into the output string. In practice, language models are commonly paired with decoding algorithms like top-k [65], top-p [102], often in combination with beam search. Various decoding heuristics and algorithms have been extensively explored to address the challenges inherent in the auto-regressive nature of language models (see Chapter 4). The decoding problem is the primary focus of Sec. 7.3.

It is important to note that employing the same language model with different decoding algorithms yields distinct outputs, thereby constituting different semantic processors. Moreover,

other factors, such as the prompting scheme, can further differentiate semantic processors. For example, the chain-of-thought [267] or least-to-most prompting [301] schemes generally result in different outputs, even when applied to the same LLM, thus defining different semantic processors.

Other semantic processors. While syntactic tokens exist for technical purposes, to support the practical computation of language models, semantic tokens are ubiquitous as they are the preferred units of human thinking. Semantic tokens are manipulated by a great variety of semantic processors, with humans being the primary example. Additionally, the tools humans build serve a purpose: to transform a meaningful input into a meaningful and useful output. Naturally, their inputs and outputs correspond to semantic tokens. Humans leverage tools like search engines, code executors, databases, APIs, etc., directly on a daily basis. There are substantial research efforts focused on developing methods for augmenting LLMs with such tools [169]. This perspective holds significance as AI systems, like any other tool, can now participate in a rich ecosystem of semantic processors that communicate via the exchange of semantic tokens.

Summary of the shift from syntactic to semantic tokens:
At the syntactic level, **syntactic tokens** are determined by the tokenizer and act as the basic symbols manipulated by language models, the **syntactic token processors**.
At the semantic level, **semantic tokens**, or **thoughts**, are meaningful units of information. **Semantic processors** are the processes that manipulate these semantic tokens. When combined with a decoding algorithm, language models become semantic processors and join the rich ecosystem of semantic processors that includes humans and tools.

7.3 Decoding: Extracting Utility from Token Processors

In the previous section, we introduce a more general definition of a token and the concept of a token processor. Now, our focus shifts to the algorithm layer responsible for orchestrating these components to solve practical tasks. We refer to this layer as the *decoding algorithm*. We draw an analogy between decoding operating on syntactic tokens and decoding operating on semantic tokens. Fig. 7.2 visually depicts this analogy.

7.3.1 General Formulation of the Decoding Problem

The objective: maximizing utility. To solve a given task, an AI system processes an input semantic token (a query) aiming to produce optimal output. This is where the concept of a *utility function*, denoted as $u_t(\mathbf{y}^\sigma|\mathbf{x}^\sigma)$, becomes critical. It scores candidate semantic token outputs, assessing how effectively they solve the task for the specific input $\mathbf{x}^\sigma$. For instance,

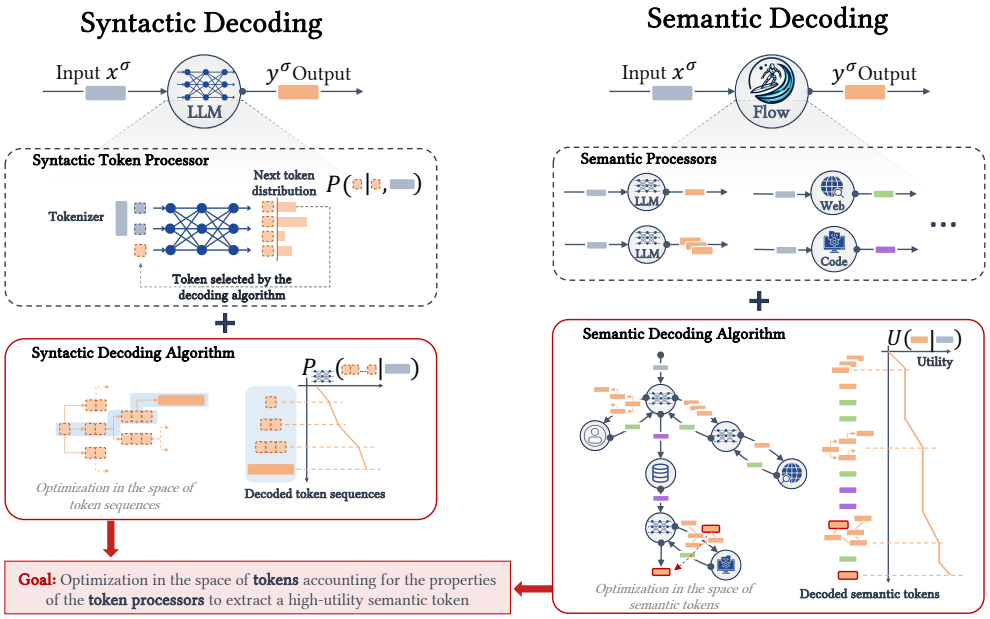


Figure 7.2: **Illustrating the analogy between syntactic and semantic decoding perspectives** On the left side, we depict the conventional (syntactic) decoding process of generating sequences of tokens from an auto-regressive language model. The decoding algorithm strategically uses the language model to produce a high-utility output sequence. Similarly, we frame recent advancements in AI, human, and tool collaboration as the semantic-level analogy of this process. Here, the computational units are referred to as *semantic processors*, which manipulate semantic tokens, representing semantically coherent units. The semantic decoding algorithm harnesses the capabilities of the semantic processors, orchestrating their computation through semantic token exchange to extract a high-utility output. Both perspectives share a common objective: extracting high-utility output by leveraging the token processors available during inference.

in machine translation, the utility function might judge how well the translation $\mathbf{y}^\sigma$ retains the original meaning conveyed by $\mathbf{x}^\sigma$. In an ideal scenario, when presented with input $\mathbf{x}^\sigma$, the system selects the output with the highest utility score: $\text{argmax}_{\mathbf{y}^\sigma \in \Gamma} u_t(\mathbf{y}^\sigma|\mathbf{x}^\sigma)$.

In practice, the AI system does not have access to the utility function during inference. However, it can rely on value models, which estimate the utility function, to guide the decoding process effectively.

The decoding problem. AI systems process information by manipulating tokens using specialized computational components, the token processors. To solve a given task effectively, a dedicated decoding algorithm coordinates the interactions between these token processors—carefully orchestrating the computation based on their individual properties, capabilities, and limitations to reliably extract high-utility outputs. At the syntactic level, the syntactic decoding algorithm utilizes syntactic token processors (language models) to manipulate syn-

tactic tokens. By analogy, we propose to conceptualize the orchestrated collaboration between AIs, humans, and tools as the semantic equivalent, wherein the semantic decoding algorithm utilizes semantic processors to manipulate semantic tokens. Both algorithms perform optimization within the token space with the objective of extracting high-utility tokens.

7.3.2 Syntactic Decoding

The goal of syntactic decoding algorithms is to retrieve an element maximizing a given utility function from the space of possible token sequences. Syntactic decoding algorithms rely on the probability distribution induced by a language model as a primary source of signal and aim to retrieve a sequence that maximizes the model's likelihood. However, this approach faces two main challenges. Firstly, the optimization problem $\text{argmax}\, y \in \mathscr{Y}\, p\theta(y;|;x)$ becomes intractable due to the exponentially large state space, necessitating approximation techniques. The commonly employed strategies to tackle this difficulty rely on greedy heuristics, such as beam search [245], which focuses on the most probable tokens, either deterministically selecting the top-k candidates or sampling from the top-k or top-p tokens from the distribution [65]. Second, as demonstrated in Chapter 4, outputs associated with high likelihood are not necessarily of high utility, and effective decoding algorithms need to account for the potential misalignment. Methods to address this problem include (i) value-guided beam search, which uses a greedy strategy similar to beam search but selects the next token using a linear combination of the model's likelihood and the scores from a value model; and (ii) Monte Carlo tree search (MCTS) [29], which allocates a fixed computation budget for an informed exploration of multiple paths in the decoding tree before token selection. See Chapter 4 for a more comprehensive overview of prior work on syntactic decoding from an optimization perspective.

7.3.3 Semantic Decoding

We now discuss the proposal of *semantic decoding*. In this perspective, the objective remains the same as that of syntactic decoding: generating a high-utility (semantic) token. However, the fundamental unit of information shifts from syntactic tokens to semantic ones, and the basic computational processes shift from syntactic processors to semantic ones.

A semantic decoding algorithm coordinates the exchange of semantic tokens among semantic processors to navigate through the semantic token space and identify a high-utility trajectory in semantic space.

Chain-of-thought (CoT) is a fundamental example, which generates a sequence of thoughts (semantic tokens) before arriving at an answer. In the initial version, the semantic tokens in the chain do not interact with any other semantic processors or algorithmic components. However, numerous variants have quickly emerged, including: (i) Sampling and combining multiple reasoning chains [262], (ii) Incorporating feedback from other models, symbolic

tools, or human inputs, [263, 264, 78, 190, 118], (iii) Exploring non-linear trajectories in semantic space, such as trees [287, 153, 282] or graphs [17, 289], and (iv) Utilizing evolutionary algorithms to select promising semantic tokens, e.g., FunSearch [212] or PromptBreeder [66].

The syntactic decoding setup often operates under the constraint to produce the full trajectory of syntactic tokens as its output. In contrast, semantic decoding has greater flexibility to manipulate its trajectories and select only a subset of the trajectory as the final output, often selecting its last semantic token as the output. The constraint to output the entire trajectory makes every decision critical for the quality of the answer. This also renders value estimation more challenging because the value model has to estimate the expected utility for partial outputs that may not look like a candidate's answer and often are not even meaningful. However, at the semantic level, the semantic decoding algorithm has the flexibility to backtrack and meaningfully remove parts of the trajectory. It's worth noting that, in principle, nothing prohibits the syntactic decoding algorithm from producing subsets of the trajectory as the output. Recent examples include the usage of pause tokens, which are automatically removed from the final output but used during decoding to allow the model more computational steps.

In Sec. 7.4, we explore how adopting the semantic decoding perspective, which focuses on the optimization and search performed in the semantic space, enables us to broaden our understanding of what is possible for orchestrated interactions.

7.3.4 Connections with Pragmatics and Semiotics

Pragmatics, a subfield of linguistics, provides a compelling framework for understanding semantic decoding [21]. Just as pragmatics views communication as a goal-oriented process where meaning emerges through dynamic interactions between participants, semantic decoding orchestrates purposeful exchanges of semantic tokens between processors. This parallel reveals semantic decoding as fundamentally pragmatic in nature, with each token exchange shaped and optimized by both context and intended outcomes.

The semantic decoding perspective also intersects with semiotics [32], the study of signs, symbols, and their interpretation. By bridging syntactic and semantic tokens, language models bring meaningless symbols into a space where semantic decoding algorithms can interpret and manipulate these concepts based on their pragmatic usage from various other semantic processors.

By focusing on the optimization of utility through the interaction of semantic processors, the semantic decoding perspective encompasses both the semiotic notion of meaning emergence and the pragmatic emphasis on the context-dependent nature of communication in achieving specific goals.

7.3.5 Flows as Semantic Decoding Algorithms

The semantic decoding perspective we propose emerges from *Flows*, the abstraction for modeling structured interactions among AI systems, tools, and humans, proposed in Chapter 6. This abstraction revolves around Flows as compositional, self-contained, goal-driven entities that execute a semantically meaningful unit of work and communicate solely through semantic tokens. Flows represent semantically meaningful blocks of computation performed either by directly utilizing a "tool," as in Atomic Flows, or from purposeful interactions among other Flows, as in Composite Flows. Crucially, in the *Flows* abstraction, the notion of a *tool* is general and encompasses everything from a database, a compiler, or a search engine to powerful AI systems like LLaMA [253], Stable Diffusion [210], and GPT-4 [186]; or a human. Fundamentally, the concept of an Atomic Flow lifts all tools into a single semantic space shared with Composite Flows, in which they can all communicate via semantic tokens (*i.e.*, messages).

Flows *are* semantic decoding algorithms as they orchestrate semantic processors to produce a useful semantic token as output. The semantic decoding perspective is concerned with the optimization that Flows performs in the semantic space (the *what*), whereas the framework of *Flows* is the conceptual tool enabling the design and implementation of these compositional interactions (the *how*). Notably, a semantic decoding algorithm—implemented by a Flow—becomes a black-box engine that reads a semantic token as input and produces a semantic token as output, therefore becoming a semantic processor itself for other Flows to utilize. This compositional property of Flows enables open-ended complexity growth by constantly increasing the pool of existing semantic processors, blurring the frontier between a semantic decoding algorithm and a semantic processor. A Flow can be both a semantic processor within a larger computation or a semantic decoding algorithm when it is itself the outermost algorithmic layer under scrutiny for the purpose of understanding what type of optimization it is performing to construct its output semantic token. Fig. 7.3 illustrates the compositional properties of *Flows* and its close connection to semantic decoding through an example. Throughout the rest of the chapter, we use the terms Flow and semantic decoding algorithm interchangeably.

Beyond the benefits at the conceptual level, the principled abstraction of *Flows* and its accompanying library aiFlows enable seamless asynchronous and distributed execution, which are necessary for many promising directions, as discussed in Sec. 7.4 and Sec. 7.5.

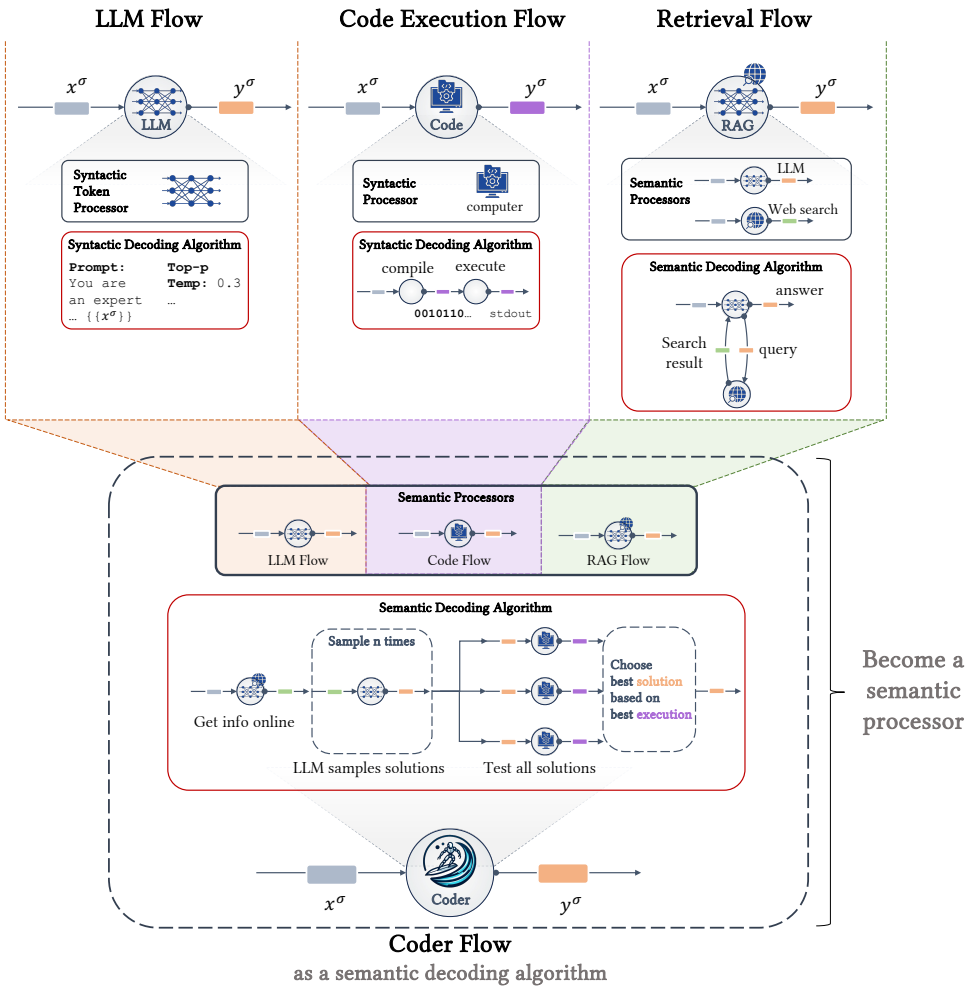


Figure 7.3: **Illustrating the compositionality of Flows.** On the top, we showcase three semantic processors. The first one is implemented using a prompted language model, the second is a code executor wrapped in a Flow to enable communication, and the last one is a semantic decoding algorithm itself (Retrieval Augmented Generator), orchestrating both an LLM and a search engine. Together, these three semantic processors are orchestrated as part of a new Flow, producing a semantic decoding algorithm designed to solve coding problems. In this example, the Retrieval Flow initially extracts useful information from the web. Subsequently, the LLM Flow samples many candidate solutions, each of which is executed and tested by the code execution Flow. Finally, the answer to be returned is selected. This programmer Flow can then serve as a semantic processor for others to utilize. Flows, implementations of semantic decoding algorithms, become semantic processors themselves as they read and generate semantic tokens. They can then join the pool of semantic processors available for other Flows to utilize. This enables an open-ended complexity growth, with modular building blocks seamlessly interfaced through communication via semantic tokens. The *Flows* abstraction underscores that the distinction between semantic processors and semantic decoding algorithms is only in perspective.

Summary of the decoding perspective:
A **decoding algorithm** is an algorithmic layer on top of token processors that orchestrates the computation over tokens to extract a high-utility output. A **semantic decoding** algorithm, in turn, is a decoding algorithm orchestrating semantic processors manipulating semantic tokens. It represents the semantic counterpart to the well-known problem of syntactic decoding from auto-regressive language models.
A semantic decoding algorithm is itself a semantic processor, ready to be employed by other, more complex, semantic decoding algorithms. This compositional property, emerging at the semantic level, enables **open-ended** complexity growth. The compositonal ***Flows*** framework capitalizes on this property to enable and facilitate the modeling, implementation, and systematic study of arbitrarily complex structured interactions between tools, models, and humans. Then, certain equivalences come to light: *Flow* $\cong$ *semantic decoding algorithm* $\cong$ *semantic processor*. *Flows* provides the infrastructure for engineering semantic decoding algorithms.

7.4 Semantic Decoding: Optimization in the Semantic Space

By adopting the semantic decoding perspective, we can analyze the orchestrated interactions between AI, tools, and humans based on the optimization processes they perform and the heuristics they use to navigate the semantic space. When compared to the syntactic level, the semantic level offers distinct advantages that semantic decoding algorithms can leverage. These advantages include:

Effectiveness in exploration: The semantic space induced by semantic processors inherits meaningful structure from the underlying tools and operations it represents. This intrinsic structure makes the semantic space naturally suited for utility optimization. In contrast, syntactic decoding faces a fundamental challenge: semantic concepts can manifest in numerous syntactic forms, making optimization in syntactic space inefficient and disconnected from semantic meaning. Moreover, semantic decoding algorithms can explore the output space non-linearly, unrestricted by the autoregressive constraints of syntactic decoding. This combination of meaningful structure and flexible exploration enables substantially more effective search strategies in the semantic space.

Human interpretability: Semantic tokens carry meaningful information, allowing humans to fully understand the computation and optimization performed by the semantic decoding algorithm. As semantic processors themselves, humans can both monitor and seamlessly engage in these optimization processes directly.

Open-endedness: Semantic decoding algorithms operate over semantic tokens—they read a semantic token as input and produce a semantic token as output, which renders them semantic processors. Therefore, they can be readily integrated within the computation of

another larger semantic decoding algorithm with their internal optimization abstracted away as a black-box semantic processor. This compositional property is critical for enabling the design and implementation of computational processes with unbounded complexity, making the framework inherently open-ended.

The design space for semantic decoding algorithms is vast, and we are just scratching the surface of its full potential. As a community, we still haven't properly formalized the problem and lack methods for systematically discovering, crafting, and learning semantic decoding algorithms. In this section, we categorize the types of search and optimization that can be performed in the semantic space, drawing connections to concepts from syntactic decoding. In Sec. 7.5, we explore opportunities for future research.

7.4.1 Heuristic Decoding Patterns: Grammars of Thoughts

A simple strategy for navigating the semantic space is to rely on predefined workflows dictating which types of semantic tokens should be generated at which moments. This programmed exchange of semantic tokens among semantic processors is designed to ensure a robust progression towards a high-utility final output. This line of research has been initiated by the Chain-of-Thought (CoT) method [267], a prompting method enforcing the generation of intermediate semantic tokens before producing a final output.

Several works have studied more sophisticated reasoning patterns, incorporating additional semantic processors. For example, ReAct employs a two-step approach of switching between a reasoning and an acting step. This allows the model to leverage external tools to generate intermediary semantic tokens. Some research splits the problem-solving process into two phases implemented by dedicated planning and acting Flows [263, 264, 78, 190, 118], or leverage feedback-giving Flows [270, 190, 118]. In Chapter 6, we demonstrated the potential of these patterns while highlighting the necessity of more systematic evaluation and research comparing diverse interaction patterns across tasks.

An interesting analogy can be drawn with the concept of constrained decoding in the syntactic decoding literature, which we study in Chapter 3. Constrained decoding refers to methods enforcing constraints during inference time [255, 85]. For instance, one may want to prevent repeated n-grams [239, 6], or enforce predefined structural constraints on the output [101, 108, 199, 119, 57, 231], expressible with formal grammars [221, 213, 85].

Then, it is worth contemplating what the semantic equivalent of a grammar constraint would be. Such a *grammar of thoughts* would represent formal constraints on trajectories' structure in the semantic space, akin to generalized reasoning patterns. Planning before acting, validating with external tools before answering, and attempting to refute one's own answer would be simple examples of such grammar.

7.4.2 Meta-Heuristics: Sampling and Value-Guided Search in Semantic Space

At the syntactic level, various decoding strategies exploit the probabilistic nature of language models, including top-p [102], top-k [65], and stochastic beams [129, 161], using sampling of tokens to better explore the space of possible output sequences.

Given that language models are also probabilistic generators of semantic tokens, exploration of the semantic space through sampling naturally arises. Many existing semantic decoding algorithms are built on the foundation of sampling semantic tokens. For instance, self-consistency [262] extends CoT by sampling multiple chains, thereby exploring different reasoning paths. This concept is further expanded to non-linear chains with approaches like Tree-of-Thought [287, 153, 282], and Graph-of-Thoughts [17, 289].

To ensure navigation towards high-utility regions of the output space, complementing sampling with an external signal, such as a value model—an estimator of the final utility given the current trajectory—can often be beneficial. At the syntactic level, this idea is embodied by Value-Guided Beam Search (VGBS) [95, 206, 130], a variant of beam search utilizing a value model in addition to the model's likelihood to decide which next token to sample. Monte-Carlo tree search (MCTS) [29, 120] extends this idea by performing simulations to explore the utility of future outcomes before making the next decision. In Chapter 4, we observed that, at the syntactic level, value-guided and simulation-based decoding methods tend to be prohibitively expensive, especially for LLMs. In contrast, semantic-level decoding operates on far fewer tokens than syntactic decoding, thereby significantly reducing the number of required value estimates and making these ideas computationally feasible even for LLMs.

Additionally, at the semantic level, the idea of a value model can be implemented very flexibly by leveraging the rich ecosystem of semantic processors. These processors offer extensive possibilities for incorporating external signals in the computation. Language models prompted to perform different roles or with different side information can provide feedback by estimating the value of a current stream of semantic tokens [283, 232, 190]. Tools or humans with access to external information, acting in the world, or executing code can further provide reliable grounded estimates of utility [78, 39, 118, 141]. Already, Ding et al. [59] hint at the idea of performing MCTS-based planning in semantic space.

Semantic decoding algorithms like FunSearch [212] and PromptBreeder [66] utilize a language model as the sampler of candidate solutions (semantic tokens) within evolutionary algorithms optimizing a population of candidate solutions. The fitness function can be viewed as a simple value model guiding the decoding of the next semantic tokens, *i.e.*, the next candidate solutions. These examples showcase the effectiveness of combining optimization through search based on sampling from a competent sampler, *i.e.*, a language model.

Currently in its early stages, this research direction lacks proper formalization within the optimization domain. Nevertheless, it shows great potential to enable the emergence of new types of AI capabilities, as evidenced by systems such as FunSearch [212].

7.4.3 Learning the Flow: Embracing Optimization in Semantic Space

Moving beyond heuristics and meta-heuristics, one could fully commit to the optimization perspective within the semantic space. At a micro level, optimization algorithms could focus on enhancing performance by training the semantic processors to collaborate more effectively. We term this approach as *learning to collaborate*. Alternatively, taking a more holistic view of the problem, optimization might involve training a controller that determines which semantic processor to invoke at each time step along with the appropriate parameters. We refer to this approach as *learning to orchestrate*. For example, one could utilize reinforcement learning [12] or reward-based supervised learning [183, 202, 91] to backpropagate a learning signal through semantic tokens, optimizing over sequences of syntactic tokens directly. Systems such as AutoGPT [209] or BabyAGI [177] already use a prompted language model as a heuristically optimized controller of a general-purpose semantic decoding algorithm. Moreover, learning to orchestrate can be seen as a planning problem, and therefore, ideas from the extensive literature on planning could be explored in this context.

This general notion of learning the semantic decoding algorithm builds upon previous work at the syntactic level. For instance, Wiseman and Rush [273] introduced a differentiable relaxation of beam-search, and Collobert, Hannun, and Synnaeve [45] developed a fully differentiable beam-search that can be optimized during training.

Summary of semantic decoding optimization:
Decoding directly in the semantic space offers many benefits, including flexible and effective exploration of the meaningfully structured semantic space, human-interpretable computation, and open-endedness enabled by the compositional nature of Flows.
The optimization performed in the semantic space can be broadly categorized into three main types: (i) **Heuristic decoding patterns**: programmed interactions such as CoT, ReAct, or meta-patterns like planning before acting and iterative, feedback-based refinement. (ii) **Sampling and value-guided search**: interactions defining optimization strategies that explore the semantic space by strategically sampling semantic tokens and leveraging a value function to guide the process. (iii) **Learning to optimize in the semantic space**: *learning to decode* by training the semantic decoding algorithms and their components. For instance, learning to collaborate, orchestrate, search, discover effective reasoning patterns, and more.

7.5 Research and Application Opportunities

In the previous section, we highlighted the distinctive benefits of the semantic decoding perspective and outlined what kind of optimization and search can be performed in the semantic space. We conclude this chapter by shifting our focus to research and application opportunities emerging from the work presented in this thesis.

7.5.1 Prompt Engineering and Meta-Prompt Engineering

Prompt engineering played an important role in the remarkable success of LLMs across fields [24, 127]. For example, compared to traditional prompting methods, CoT [267] has demonstrated a threefold increase in performance on the GSM8K dataset. Likewise, for competitive coding, a fixed collaboration pattern between two GPT-4 instances—a coder and a critic with access to a code executor—increases the solve rate by 1.8 times, as we saw in Chapter 6. These represent just a glimpse of the many examples where the performance of LLM-based approaches has been significantly improved without fine-tuning. Researchers continue to push boundaries by crafting complex patterns to tackle increasingly complex tasks [212] and exploring meta-prompting methods [66]. Viewing (meta-)prompt engineering as optimization in semantic space opens the door to novel ideas and provides a principled structure for the research efforts in this particularly active and promising direction.

7.5.2 Synthetic Data Generation Flows

Data, itself composed of semantic tokens, can become the focal point for Flows dedicated to manipulating or synthesizing data. The synthetic data generation Flows can leverage domain knowledge [249], task properties [156, 258, 117], or collaboration [1], and synthesize data of notably higher quality than what a single model or simple heuristics can achieve. This sets the stage for effective self-improvement loops in which a language model participates in a semantic decoding algorithm, producing high-quality synthetic data. Then, the language model improves itself through fine-tuning, thereby improving the Flow's capacity to generate even better synthetic data in a virtuous cycle [233, 25, 234, 41]. An example of such a Flow is MAGDi [36], a framework designed to distill reasoning interactions among multiple LLMs into smaller ones. This approach surpasses single-teacher distillation [144, 159] and finetuning based on reasoning trajectories sampled from GPT-4 [34].

7.5.3 Human in the Loop and Human-Computer Interaction

Humans operate within the semantic space and can seamlessly integrate into semantic decoding algorithms as just another type of semantic processor. This integration creates numerous opportunities to explore diverse human–computer interactions within a principled optimization-based framework. This framework can leverage human cognition as a computational input for semantic decoding algorithms aimed at maximizing a utility function. For instance, humans can provide nuanced low-level feedback, offer high-level guidance—as in our experiments in Chapter 6—or simply observe the ongoing exchange of semantic tokens [118, 26, 281, 123, 145]. Moreover, humans can dynamically switch roles during the execution, intervening, pausing the process, and resuming as necessary. Flow engineering methods, such as searching for heuristic reasoning patterns (Sec. 7.4.1), crafting guided search methods (Sec. 7.4.2), or learning the Flow (Sec. 7.4.3), can be designed to optimize objectives that

involve maximizing a utility function while minimizing the cognitive cost for the human in the loop [106].

7.5.4 General AI Assistants

Considerable research efforts are currently focused on constructing general-purpose assistants. Existing methodologies involve utilizing LLMs as controllers equipped with selected tools to enhance abilities [177, 209, 264, 279]. However, recent evaluations on benchmarks for agents have revealed a notable performance gap compared to humans, highlighting the need for improvements [116, 31]. For instance, on the GAIA benchmark [170], GPT-4, when equipped with tools, achieves only 15% accuracy, while humans attain 92% accuracy. Examining this challenge through the perspective of semantic decoding, it can be framed as the development or learning of a general-purpose semantic decoding algorithm underpinning the agentic AI system.

7.5.5 Evaluation and Diagnostic

Evaluating semantic decoding algorithms presents a significant challenge because they evade standard controlled benchmarks for three primary reasons. Firstly, the issue of data contamination arises because language models are trained on vast amounts of internet data, potentially including benchmark samples. To address this issue, research is needed to untangle the impact of memorization and generalization across various contexts [230, 87]. In Chapter 6, we empirically demonstrated that accounting for data contamination is essential for correctly assessing the capabilities of LLMs and, by extension, any AI system with LLM components.

Second, the dynamic nature of the environment leads to the evolution of AI systems over time. For instance, a search engine's results can vary not only due to algorithmic changes but also due to updates in the real world. Overcoming this challenge necessitates advancing diachronic evaluation methodologies specifically tailored to assess dynamic AI systems in evolving environments, as exemplified by dynamically evolving benchmarks [146].

Third, developing multi-step benchmarks for evaluating relevant AI agents' capabilities is challenging. Recently, there has been great progress in developing domain-specific benchmarks like SWE-bench and MLE-bench that focus on software engineering tasks [116, 31]. Software is inherently fully digital, which makes the development of software engineering benchmarks much more accessible and, due to its generality, has far-reaching implications across all domains. While this makes it a perfect candidate to start with, developing benchmarks for testing other relevant agentic capabilities will be necessary. Software engineering serves as an ideal initial domain for benchmark development due to its fully digital nature and inherent generality. This makes benchmark development more tractable while ensuring broad applicability across domains. However, comprehensive evaluation of AI agents will

require developing additional benchmarks that test other critical capabilities, such as complex reasoning and long-term planning.

Despite these evaluation challenges, it remains imperative to develop methods that effectively discern between effective and ineffective semantic decoding algorithms. This understanding is crucial for informing users about the expected behavior of AI systems and enabling practitioners to enhance system performance. Tools from causality, such as causal mediation analysis, can be particularly useful in recognizing which components or messages were critical to the computation's success or failure. These diagnostic considerations are tightly linked to the interpretability challenges discussed below.

7.5.6 Interpretability, Control, and Error Mitigation

Interpretability. A semantic decoding algorithm, as any AI system, is subject to questions of explainability to understand *why* some outputs were produced [274, 200, 149]. One approach is to perform behavioral analyses by manipulating inputs and observing the resulting effects on the outputs. Additionally, model-agnostic feature importance methods, such as LIME [208] or SHAP [292], can be expanded to include semantic tokens rather than just syntactic features. Opening the black box to inspect the network of semantic tokens exchange leads the path toward mechanistic interpretability of semantic decoding algorithms. This involves manipulating intermediate semantic tokens, placing the system in a counterfactual state, and then resuming the computation to precisely measure the impact of each computational step [191, 84]. In general, interpreting semantic decoding algorithms is somewhat simpler than language models because semantic tokens serve as discrete, semantically meaningful bottlenecks in the computation that can be easily inspected and intervened upon.

Control and ethics. Closely linked to the issue of explainability is the notion of control [274, 191]. By breaking down computation into discrete, human-interpretable chunks, we obtain many levers for controlling the computation. This can be done manually by humans or automatically by appending dedicated semantic processors to inspect, correct, or prohibit some predefined undesirable intermediate steps, thus preventing undesired outcomes. For example, specialized *ethics semantic processors* (possibly complex Flows themselves) can be inserted at critical steps to exclude unwanted semantic tokens and steer the computation toward areas of the output space that align with predefined ethical objectives [74]. Such ethics components might leverage expert debiasing methods that scrutinize semantic tokens for potential societal biases and either remove them by reformulating or providing informed feedback.

7.5.7 Richer Semantic Spaces

From syntactic to semantic and back. One advantage of semantic tokens is that they are readily understood by humans, rendering semantic decoding algorithms inspectable, interpretable, and conducive to human participation. Yet, the decoding perspective does not inherently require semantic tokens to be human-interpretable. One could imagine a collaboration between LLMs and tools that, for the sake of communication efficiency or effectiveness, relies on new made-up languages. These new semantic spaces can be learned as part of the *learning the flow* pipelines to optimize the amount of useful information exchanged per message.

Multimodal semantic tokens. Another way in which the concept of semantic token can be stretched is by considering other modalities. The semantic decoding perspective easily accommodates more general informational units [204]. Developing more competent multimodal language models is an active area of research [275]. Multimodal models can readily participate in semantic decoding algorithms, significantly enlarging the semantic space and the variety of signals available to guide exploration toward high-utility outputs.

7.5.8 Infrastructure

To support the innovations discussed above, it is imperative to have robust infrastructures. Significant efforts have already been invested in creating efficient abstractions [154, 142, 229, 33, 103, 278]. However, the *Flows* framework stands out by introducing the modularity and compositionality essential for systematically constructing complex systems (Sec. 6.3). Its support for concurrent and distributed execution unlocks creative applications like FunSearch [212], PromptBreeder [66], meta-reasoning Flows [118], as well as arbitrary peer-to-peer collaborations between Flows, which could define anything from a stateless tool to a general assistant or autonomous agent.

Beyond proper abstractions, *Flows* and semantic decoding algorithms enable a level of complexity and flexibility that comes with additional technical challenges. For instance, *Flows* supports open-ended complexity and allows practitioners to freely develop and publicly deploy Flows, thereby necessitating systematic and contextual Flow indexing and retrieval (*i.e.*, a search engine over Flows). Additionally, executing Flows may depend on resource-intensive semantic processors, such as commercial LLMs, necessitating efficiency optimizations to minimize resource usage or latency while maintaining utility.

Flow indexing and retrieval. Similar to how a web page provides information or access to services, a Flow provides access to meaningful computation. Given the ability to publicly deploy Flows and the infrastructure for peer-to-peer communication between Flows, the need for a search engine over the space of Flows will grow as the number of publicly accessible Flows increases. Developing scalable yet effective methods for indexing Flows (*i.e.*, semantic

computation), akin to PageRank for web pages, is bound to become an important research direction in the near future.

Efficiency optimization. Improving the efficiency of a Flow can be achieved by enhancing the efficiency of its semantic processors. Thankfully, a rich body of work is dedicated to improving the efficiency of language models' inference. Techniques like batching and key-value caching can mitigate the cost and latency of decoding long sequences autoregressively [198]. Speculative decoding methods, which involve generating candidate samples using smaller models and then refining or filtering these samples with larger models, represent a different approach to optimization [222, 35, 124, 240]. Further optimizations include memory usage reduction through weight quantization [290, 55, 71, 56, 280, 11], hardware-aware algorithmic advancements like FlashAttention [49], and leveraging non-homogeneous computational costs in the transformer graph, as exemplified by SkipDecode [48] and PowerInfer [237].

At the semantic level, caching Flow calls with soft, approximate caches [203] and dynamically routing queries to different models based on their properties [107, 152, 218, 294, 138] are feasible strategies. Concurrent execution of semantic processors, such as Skeleton-of-Thoughts [182], also contributes to latency gains. Despite these promising approaches, the concept of *speculative semantic decoding* remains relatively unexplored. Systematic studies could explore replacing expensive sub-flow components with faster, more economical modules trained to emulate or cache computations from costly sub-Flows.

8 Conclusion

In this thesis, we study the abstractions, methods, and infrastructure needed to enable and support the development of compound agentic AI systems in the era of large language models (LLMs). The thesis addresses fundamental challenges at every stage of the design of such a system.

First, we examine the integration of LLMs into two-component collaborations without modifying the model. Chapter 3 shows how constrained decoding can be used to ensure that the LLMs' outputs follow explicitly specified schemas—necessary for adherence to interfaces in a collaboration. We empirically demonstrate the effectiveness of our approach on the task of closed information extraction (cIE), the problem of extracting structured semantic information from unstructured texts, which is well-studied in the field of IE and exemplifies the challenge. Our end-to-end autoregressive approach, paired with contained decoding, achieves state-of-the-art performance, generalizes from fewer training data points, and scales to a previously unmanageable number of items in the knowledge base. Chapter 4 generalizes this idea and studies decoding as a tool for alignment more broadly. Concretely, we argue that the misalignment between the model's likelihood and the task-specific notion of utility is the key factor to understanding the effectiveness of decoding algorithms, and introduce a taxonomy of misalignment mitigation strategies (MMSs). The MMS taxonomy groups decoding algorithms based on their implicit assumptions about likelihood–utility misalignment, yielding general statements about their applicability across tasks. By analyzing the correlation between the likelihood and the utility of predictions across a diverse set of tasks, we provide empirical evidence supporting the proposed taxonomy and a set of principles to structure reasoning when choosing a decoding algorithm.

Second, we turn to scenarios where the underlying model's capabilities are insufficient for effective collaboration, but the training signal necessary for fine-tuning is not readily available. To address this challenge, Chapter 5 shows that useful data can be synthetically generated even for tasks that cannot be solved directly by LLMs: for problems with structured outputs, it is possible to prompt an LLM to perform the task in the reverse direction, by generating plausible input text for a target output structure. We empirically demonstrate the effectiveness of this approach on a task where collecting ground-truth data is challenging, and no

satisfactory dataset existed prior to our work. Crucially, leveraging this asymmetry principle in task difficulty makes it possible to produce large-scale, high-quality data for any complex tasks for which we can come up with an (inverse) formulation that the LLM can address more effectively.

Third, we focus on enabling the design and implementation of collaborations among AI systems, tools, and humans. Chapter 6 proposes *Flows*, an abstraction that, in concert with the accompanying library `aiFlows`, provides a theoretical and practical infrastructure with modular and concurrency-friendly design, which enables the modeling, implementation, and systematic study of arbitrarily complex structured interactions. In a thorough investigation of core interaction patterns on the task of competitive coding, we empirically demonstrate the potential of Flows, the necessity of more systematic research, and the value brought by *Flows* and `aiFlows` in support of these research efforts.

Fourth, we examine the exploration of the vast space of structured interaction patterns. Chapter 7 proposes semantic decoding, which formalizes LLMs, humans, and other tools as semantic processors that read and generate semantic tokens, and the collaborations between them as optimization processes in the semantic space. These optimization processes correspond to semantic decoding algorithms and aim to find high-utility semantic tokens as answers to queries. This perspective allows us to systematically study the design space of semantic decoding algorithms based on the optimization they perform. By focusing on the semantic level and disregarding syntactic details, we gain a fresh perspective on the engineering of AI systems, allowing us to imagine systems with much greater complexity and capabilities.

A Appendix for Chapter 3

A.1 Additional Background and Related Work

A.1.1 Generative Open Information Extraction

Early work had focused on pipeline architecture for oIE. In general, these methods first detect the entity mentions present in the text and then, for pair of entities, in a classification setting, predict the existence of a relation between the two entities and the relation type [7, 47]. The advent of transformers [58, 135, 150] and pipeline architectures that allow for information to flow between the two subtasks – usually by sharing some parameters of the encoder – have allowed these models to do well on the oIE task. However, they do come with some general limitations: (i) assuming the existence of a single relation between a pair of entities; (ii) inability to capture the interactions between triplets.

Much of the current research is focused on studying the oIE problem in the autoregressive generative setting, which seamlessly mitigates the limitations mentioned above [111, 60, 180]. For instance, ReGen [60] significantly improves upon published results and establishes state-of-the-art results on the dataset used in the WebNLG 2020+ Challenge [28]. REBEL [111], on the other hand, achieves state-of-the-art performances across a suite of oIE benchmarks. Moreover, both of these methods address the problem in a similar formulation that takes the text as input context and generates the output triplets token-by-token in an autoregressive fashion.

The output triplets in oIE are free-text, while cIE requires the constituent elements of the output triplets to come from the entity and relation sets associated with the KB. This makes the cIE task fundamentally harder than oIE, and renders these methods not applicable to the cIE setting.

Finally, Taillé et al. [247] make an effort to describe the many issues with the evaluation of oIE systems in literature and call for a unified evaluation setting for a fair comparison between systems. These problems get only exacerbated in cIE where the performance of a model

Dataset	Documents			Triplets			$\lvert\mathscr{E}\rvert^{\dagger}$	$\lvert\mathscr{R}\rvert^{\dagger}$
	training	validation	test	training	validation	test		
REBEL	1,899,331	104,960	105,516	5,147,836	284,268	284,936	1,498,143	857
Wiki-NRE	223,536	980	29,619	298,489	1,317	39,678	278,204	157
Geo-NRE	–	–	1,000	–	–	1,000	124	11
FewRel	–	26,892*	27,650	–	26,892*	27,650	64,762	80

Table A.1: **Statistics of the datasets.** †With an abuse of notation here we indicate the amount of unique entities and relations for each dataset and not the size of the Knowledge Base associated with it (see Section 3.4 for more details). * Note that we do not use the validation FewRel data in our experiment, but we release this split as well.

would highly depend on the entity and relation catalogue considered. To alleviate some of these issues, we annotate the REBEL dataset [111] with unique textual entity identifiers and textual relation labels, and propose a suite of meaningful evaluation settings while considering an approximately 6 million long entity catalogue comprised of all the entities in the English Wikipedia, and 857 long relation catalogue supported by the dataset.

A.2 Datasets

Table A.1 summarizes the statistics of all datasets used in this work. For each dataset, we remove datapoints containing triplets with entities that do not have an associated Wikipedia page (*i.e.*, entities not associated to a unique name). This filtering removes a negligible portion of the data in most cases (*i.e.*, <0.5%) except for REBEL where 3.4% of datapoints were removed.

We evaluate the models in standard setups for Wiki-NRE and Geo-NRE. For these datasets, the schema is unrealistically small: ≈300K entities with 157 relations for Wiki-NRE and 124 entities with 11 relations for Geo-NRE. Therefore, we scale to previously unexplored schema sizes for cIE using the REBEL dataset (≈6M entities and 857 relations). We also use FewRel as a high-quality dataset for recall evaluation using the large schema from REBEL.

REBEL [111] is a dataset created from Wikipedia abstracts. It consists of an alignment between sentences, Wikipedia hyperlinks and their corresponding Wikidata entities, and relations. REBEL proposed an alignment expanding on Elsahar et al. [63], a pipeline of mention detection, coreference resolution, entity disambiguation and then mapping triplets to each sentence. Huguet Cabot and Navigli [111] further filtered false positives using a natural language inference model to check if the relation was truly entailed by the text. In this setting, we consider the full ≈6M long entity and 857 long relation catalog. We use this dataset for both training and testing. Additionally, we employ REBEL to analyze the performance as a function of the number of relations, by simulating different environments pertaining to subsets of the top-n most frequent relations.

Wiki-NRE [254] is a dataset created from Wikipedia. Authors aligned hyperlinks to Wikidata entities as in REBEL but they applied a different filtering: they (i) extracted sentences that contain implicit entity names using co-reference resolution [44], and (ii) they filtered and assigned relations to sentences using paraphrase detection from different sources [179, 76, 89]. We used this dataset for both training and testing.

Geo-NRE [254] is constructed in the same way as Wiki-NRE but from a collection of user reviews on 100 popular landmarks in Australia, instead of Wikipedia. Due to its small size and to compare with the literature, we used this dataset only for testing.

FewRel [93] is also extracted from Wikipedia where Wikidata is the KB. Contrary to the other datasets, FewRel does not provide distant supervision but it is fully annotated by humans. The dataset was first automatically constructed and then filtered as annotators were asked to judge whether the relations are explicitly expressed in the sentences. Each input in FewRel is associated with a single triplet only, and not all of the triplets entailed by it. Therefore, this dataset can be used for precisely measuring recall (but not precision or F1). We employ it only for testing. To simulate a more realistic scenario, we train the models on many relations, and leverage the high quality FewRel data to calculate the performance metrics for the subset of relations annotated.

A.3 Performance Metrics

We measure standard precision, recall and F1 for all settings. A fact is regarded as correct if the relation and the two corresponding entities are all correct. More precisely, we denote the set of all predicted triplets of a document $d \in \mathscr{D}$ as P_d, and the set of gold triplets as G_d. Then:

$$\text{micro-precision} = \sum_{d\in\mathscr{D}} |P_d \cap G_d| \Big/ \sum_{d\in\mathscr{D}} |P_d| \,, \tag{A.1}$$

and

$$\text{micro-recall} = \sum_{d\in\mathscr{D}} |P_d \cap G_d| \Big/ \sum_{d\in\mathscr{D}} |G_d| \,. \tag{A.2}$$

Micro scores are useful for measuring the overall performance of a model but they are less informative for imbalanced datasets (*e.g.*, when some entities or relations are disproportionately more present in both training and test sets). Indeed, micro scores assign equal weight to every sample while macro scores assign equal weight to every class. Thus, we also measure macro scores by aggregating per relation type. If we denote $P_d^{(r)}$ and $G_d^{(r)}$ as the predicted and gold

set only containing the relation $r \in \mathscr{R}$ of a document d, then macro-precision is defined as:

$$\frac{1}{|\mathscr{R}|} \sum_{r \in \mathscr{R}} \left(\sum_{d \in \mathscr{D}} |P_d^{(r)} \cap G_d^{(r)}| \Big/ \sum_{d \in \mathscr{D}} |P_d^{(r)}| \right), \tag{A.3}$$

and macro-recall as:

$$\frac{1}{|\mathscr{R}|} \sum_{r \in \mathscr{R}} \left(\sum_{d \in \mathscr{D}} |P_d^{(r)} \cap G_d^{(r)}| \Big/ \sum_{d \in \mathscr{D}} |G_d^{(r)}| \right). \tag{A.4}$$

A.4 Note on End-to-End Baselines

We invested a considerable amount of time trying to use a strong end-to-end baseline to compare GenIE with. Unfortunately, most works do not have available or directly usable code. In particular, we first concentrated on SetGenNet [242] as, to the best of our knowledge, it is the strongest model on the task of cIE. However, the authors do not report a link to the code[1] in neither the arXiv nor the ACL Antology version of the paper. We could not find any related repository on GitHub either. For these reasons we were unable to use their method as a baseline for REBEL.

We then focused on the work most similar to SetGenNet, that is the system proposed by Trisedya et al. [254]. They released code and we were able to run it. However, the code was incomplete: they included code for training only a part of their system. They start with pre-trained word, entity and relation embeddings, but did not release code for pre-training them. The closest solution we found was using Wikipedia2Vec [284], which does not include relation embeddings. Besides, the pre-trained word embeddings on the official Wikipedia2Vec website[2] do not match the dimensionality used by Trisedya et al. [254]. Finally, the authors did not include code to train the "triple classifier" of their model. The classifier is instead directly loaded in their code. For these reasons we were unable to use their method as a baseline for REBEL.

A.5 Implementation Details

Data. The train, test and validation splits are either inherited from the original dataset (see Appendix A.2 for details) or sampled at random. To facilitate reproducibility, we release the exact splits employed in our experiments.

Additionally, we release the curated entity and relation catalogs for both the large and the small schema, in which the redirects have been resolved and each of the QID/PID is paired with a unique, semantically meaningful textual identifier. We hope that this will allow for a fair comparison of future work in which the same evaluation setup can be maintained.

[1] As of October 2021.

[2] https://wikipedia2vec.github.io/wikipedia2vec/pretrained

	Max steps	Warm-up steps	Batch size	Dropout	Weight decay	Training time
GenIE (W)	60,000	1,000	32	0.1	0.01	0.5 GPU days
GenIE (R)	100,000	5,000	384	0.1	0.01	18.5 GPU days
GenIE (R + W)	100,000	5,000	384	0.1	0.01	20.5 GPU days
GenIE - Genre	50,000	3,000	2,048	0.3	0.50	11 GPU days
GenIE - PLM	50,000	3,000	2,048	0.3	0.50	17 GPU days
SoTA Rel-class (W)	20,000	500	128	0.1	0.01	0.2 GPU days
SoTA Rel-class (R)	250,000	500	128	0.1	0.01	2.5 GPU days
SoTA Rel-class (R + W)	250,000	500	128	0.1	0.01	2.5 GPU days
SoTA Tri-class (R)	50,000	500	128	0.1	0.01	0.3 GPU days
SoTA Tri-class (W)	5,000	500	128	0.1	0.01	0.1 GPU days
SoTA Tri-class (R + W)	50,000	500	128	0.1	0.01	0.3 GPU days

Table A.2: **Hyperparameters for the different models.**

A.5.1 GenIE

Infrastructure. For training we used a single machine with 24 Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz processor cores and 441 GB of RAM, equipped with 4 Tesla V100-PCIE-16GB GPUs.

Training. The models were trained using the Adam optimizer [125] with a learning rate of 3e-5, 0.1 gradient clipping and a varying weight decay (cf. Table A.2). The learning rate is updated using a polynomial decay schedule with an end value of 0. While most of the parameters were left at their default values for BART, the rest were tuned on the respective datasets' validation set, and their corresponding optimal values are given in Table A.2.

Inference. At test time, we use Constrained Beam Search with 10 beams. We restrict the input and the output sequence to be at most 256 tokens, cutting from the right side if the input is too long. We normalize the log-probabilities by sequence length, and allow for any number of n-gram repetition. The other parameters are kept to their default values for inference with BART.

A.5.2 SotA Pipeline

We described our SotA pipeline system baseline in Sec. 3.4.3. We release code to both train and run inference with the proposed pipeline. The named entity recognition and the entity disambiguation components were not trained. The relation classification module is a linear layer on top of RoBERTa [150]. We trained it learning rate 3e-4 using the Adam optimizer [125]. We trained for a maximum number of steps using early stopping on the validation sets. We restrict the input sequence to be at most 128 tokens cutting from the right side if the input is too long. All other hyperparameters are reported in Table A.2. The triple classification module is also a linear layer on top of RoBERTa [150] with the same hyperparameters of the relation classification module but we trained for less steps.

A.6 Additional Experiments

A.6.1 Analysis of Performance as a Function of the Number of Relations

Previous works focus on small schemas meaning that few relations were considered. Indeed, classification problems on a large set of possible classes become particularly difficult under large class imbalances, which is the case here as shown by Fig. 3.2. However, scaling up to larger schemas with more relations is crucial for the models to be useful in downstream tasks. To measure the scaling ability of GenIE, we create different setups with variable numbers of relations. To create such setups, we start with the REBEL dataset and schema (857 relations) and choose subsets of relations with their associated training data. In Fig. A.1, we report GenIE and the pipeline baseline F1 for schemas with 100, 400, and 857 relations. To choose a subset of n relations, we take the n most frequent relations to mimic the strategies used by previous works to reduce the schemas [242].

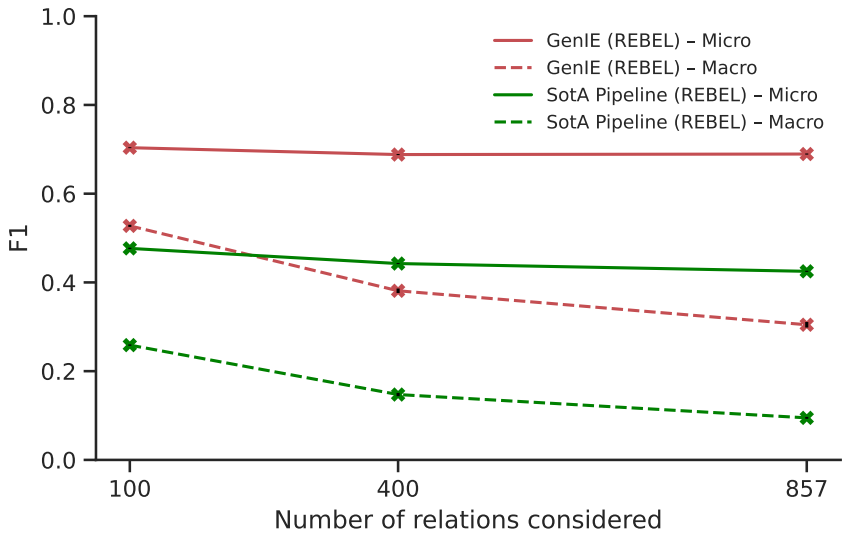


Figure A.1: **Impact of the number of relations in the schema on REBEL.** Micro and macro F1 of both GenIE and the pipeline of SotA components for 3 schema sizes: 100, 400, and 857 relations. The schema is constrained at both training and testing time. Full results (*i.e.*, precision and recall) are reported in Table A.4 in Appendix A.7.

We first observe that GenIE is always largely better than the baseline. The baseline suffers from the same difficulty as previous works; classifying among a large set of relations is hard with large imbalances. GenIE and the baseline have similar absolute decrease in performance when the number of relations increases, corresponding to a more considerable relative decrease for the baseline. More concretely, GenIE's micro F1-score goes from 70.36 % for the top 100 relations, to 68.82 % and 68.93 % for the top 400 and 857 relation setups, respectively. This translates to a relative decrease of 2 % only in the first step. For the baseline, the absolute score

of 47.67 % first falls to 44.25 % and subsequently to 42.5 % as the number of relations grows. This in turn, is an overall relative drop of almost 11 %.

Notably, when looking at precision and recall separately (*cf.* Table A.4 in Appendix A.7), GenIE has a slight proportional decrease of 1-2 absolute points, both in precision and recall, which reflects the increased difficulty of the task due to larger number of relations. The baseline exhibits a similar drop in precision, but a much more significant drop in the recall of almost 10 absolute or 16 point relative. This suggests that the baseline simply ignores most of the relations with lower occurrence counts, which is consistent with the results in Sec. 3.2, and the hypothesis that the relation classification task is a bottleneck for effectively scaling the baseline system to a large number of relations.

We already have to deploy several techniques to help the baseline better deal with these issues (see Sec. 3.4.3), while GenIE, thanks to its generative autoregressive formulation, can effectively scale and manage the inherent imbalances of the task much more naturally.

A.7 Additional Results

	Wiki-NRE			Geo-NRE		
	Precision	Recall	F1	Precision	Recall	F1
Pipeline baselines						
AIDA + MinIE	36.72	48.56	41.82	35.74	39.01	37.30
NeuralEL + MinIE	35.11	39.67	37.25	36.44	38.11	37.26
AIDA + ClauseIE	36.17	47.28	40.99	35.31	39.51	37.29
NerualEL + ClauseIE	34.45	37.86	36.07	35.63	37.91	36.73
AIDA + CNN	40.35	35.03	37.50	37.15	31.65	34.18
NeuralEL + CNN	36.89	35.21	36.03	37.81	30.05	33.49
Encoder-decoder baselines						
Single Attention	45.91	38.36	41.80	40.10	39.12	39.60
Single Attention (+pre-trained)	47.25	40.53	43.63	43.14	43.11	43.12
Single Attention (+beam)	60.56	52.31	56.13	58.69	48.51	53.12
Single Attention (+triplet classifier)	**73.78**	50.13	59.70	67.04	53.01	59.21
Transformer	46.28	38.97	42.31	45.75	46.20	45.97
Transformer (+pre-trained)	47.48	40.91	43.95	48.41	48.31	48.36
Transformer (+beam)	58.29	50.25	53.97	61.81	61.61	61.71
Transformer (+triplet classifier)	73.07	48.66	58.42	**71.24**	57.61	63.70
Our pipeline baseline	67.43	**54.22**	**60.11**	64.60	**64.05**	**64.32**

Table A.3: **Baselines comparison.** All results are taken from from Trisedya et al. [254]. Encoder-decoder baseline are proposed by the authors and other pipeline baseline include an NER and an ED system AIDA [100] or NeuralEL [128] and then a relation extraction system CNN [148], MiniE [81], or ClausIE [47]. Best results are highlighted in **bold** and second best are underlined. Our pipeline baseline scores the best or on pair among these other methods.

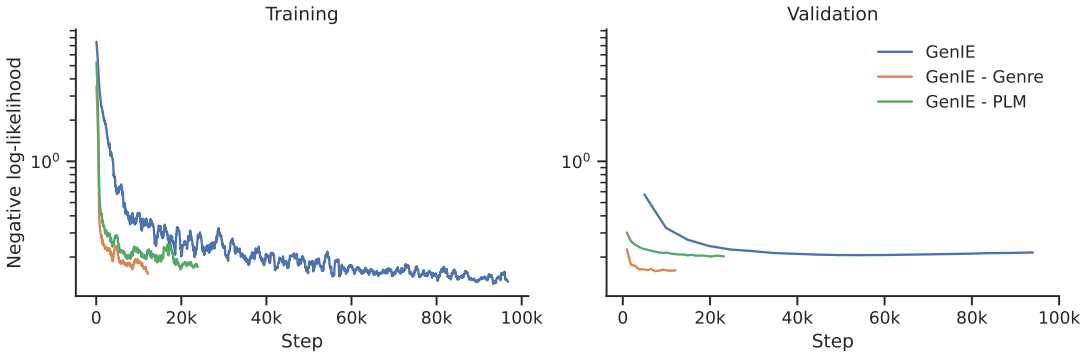


Figure A.2: **Training and validation loss curves for different initialization of our model.** GenIE starts from a random initialization, GenIE – PLM fine-tunes a BART pre-trained language model, while GenIE - GENRE is initialized with a pre-trained autoregressive entity linking model by De Cao et al. [52].

	REBEL (top 100 Relations)			**REBEL (top 400 Relations)**			**REBEL (857 Relations)**		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Micro									
GenIE	68.76 ± 0.12	72.05 ± 0.13	70.36 ± 0.10	67.10 ± 0.13	70.62 ± 0.15	68.82 ± 0.12	68.02 ± 0.15	69.87 ± 0.14	68.93 ± 0.12
SotA Pipeline	44.76 ± 0.17	50.99 ± 0.17	47.67 ± 0.16	38.98 ± 0.13	51.18 ± 0.12	44.25 ± 0.11	43.30 ± 0.15	41.73 ± 0.13	42.50 ± 0.13
Micro									
GenIE	52.26 ± 0.25	54.13 ± 0.27	52.75 ± 0.24	41.50 ± 0.66	38.53 ± 0.57	38.12 ± 0.51	33.90 ± 0.73	30.48 ± 0.65	30.46 ± 0.62
SotA Pipeline	27.41 ± 0.27	31.05 ± 0.18	25.87 ± 0.15	16.94 ± 0.63	19.00 ± 0.36	14.73 ± 0.37	12.20 ± 0.35	10.44 ± 0.22	9.48 ± 0.21

Table A.4: **Impact of the number of relations in the schema on REBEL.** The schema is constrained at both training and testing time.

B Appendix for Chapter 4

B.1 Proposed MMS Taxonomy

This section describes some of the most prominent members in each class of the proposed taxonomy.

B.1.1 Greedy Likelihood-Based Strategy

Deterministic

Greedy Search (GS). The simplest among the decoding algorithms, at each step t, GS selects the token with the highest likelihood under the model.

Beam Search (BS). An extension of GS, BS, maintains not one, but $k \in \mathbb{N}^+$ partially-decoded sequences, called beams, in parallel. At each step t, BS: (i) pre-selects the most likely k tokens for each beam; (ii) from the resulting $k \times k$ nodes, the algorithm selects the k with the highest likelihood and drops the rest.

Stochastic

An alternative that increases the diversity of output sequences is to sample the tokens at each step from the likelihood distribution $\hat{y}_t \sim p(y_t|\hat{y}_{<t}, x)$. Instead of sampling from the full distribution, these decoding algorithms typically focus greedily on the tokens corresponding to the high-probability regions.

Top-k Sampling. Top-k samples the next tokens from a truncated distribution where only the k most probable tokens are considered [65].

Top-p Sampling. Top-p samples the next tokens from a truncated distribution where only the smallest set of tokens with a probability mass bigger than (or equal to) p is considered [102].

Stochastic Beams (SB). SB samples completed outputs without replacements according to the LM's likelihood. The implementation relies on applying BS on likelihood scores perturbated with Gumbel noise [129].

B.1.2 Greedy Likelihood-Based Strategy with Pruning

Ad-Hoc Heuristics. Currently, most tasks utilize some ad-hoc heuristics. For instance, in MT, it is often necessary to discourage empty (or short) sequences by enforcing a minimal sequence length [239]. Similarly, state-of-the-art language generation models often get stuck in repetitive loops. Therefore, an n-gram repetition penalty is now part of the standard toolkit [126].

Constrained Beam Search (CBS). The idea of constraining the likelihood during decoding can be extended to include task-specific knowledge. For example, in information extraction tasks, the BS decoding strategy has been constrained to only extract outputs satisfying the predefined schema [221, 54, 119]. Then, BS only searches high-scoring outputs among the smaller subset of valid ones.

NeuroLogic. The NeuroLogic strategy enforces the satisfaction of given lexical constraints by controlling the decoding stage of sequence generation [155]. While BS aims to maximize the likelihood of the generated sequence, NeuroLogic searches for optimal output sequences among the strings that also satisfy the given constraints. Hard logic constraints are converted into a soft penalty term in the decoding objective, and beam-based search is used to find approximately optimal solutions.

B.1.3 Greedy Likelihood- and Value–Based Strategy

Value-Guided Beam Search (VGBS). It is the most intuitive example of a greedy decoding algorithm that leverages a value model [95, 206]. It uses a greedy strategy similar to BS but selects the next token using a linear combination of the LM's likelihood and the scores from the value model.

More specifically, instead of expanding each beam by the m highest-scored tokens according to the likelihood, the algorithm chooses the top m tokens according to the following scoring function:

$$s_{y_{<i},x}(y_i) = \frac{\alpha}{i}\log(p(y_{<i}y_i|x)) + (1-\alpha)v(y_{<i}y_i, x),$$

where the factor α weights the contribution of the the value model, $y_{<i}$ denotes the partially decoded sequence, and y_i corresponds to the next token under consideration.

B.1.4 Simulation-Based Strategy

Monte-Carlo Tree Search (MCTS). MCTS is the canonical example of simulation-based tree exploration informed by value. In our setup, it differs from all other decoding algorithms because, at step i, it may explore sequences of length greater than i. It is not tied to committing to local decisions without exploring the tree. In each step, MCTS has a fixed computational budget that it uses to explore multiple paths before choosing the next token. For more details, we refer to Chaffin, Claveau, and Kijak [29], whose implementation we adapt for this work.

B.1.5 Prompting-Based Strategy

Few-Shot (FS). At inference time, instead of only passing the input x, a context comprised of k examples $(x_i, y_i)_{i=1}^{k}$ is added as a prefix. The main idea is that the model will build on its semantic understanding of the relation between x_i and y_i and make the "guided" likelihood better aligned with the utility [24].

Chain-of-Thought (CoT). The CoT decoding method [266] is a conceptual extension of FS which presents the examples' targets as a sequence of steps that lead to the solution. This format is particularly helpful for tasks that require multi-step reasoning, with which transformers generally struggle.

B.2 Experimental Setup

This section provides additional details about the experimental setup.

B.2.1 Details about Data, Models, and Utility Functions

In Table 4.1, we present a summary of the tasks, utility functions, misalignment types, model, and dataset. We now give a brief description of each task:

Closed Information Extraction (cIE) with the REBEL dataset [111] and the GenIE model proposed in Chapter 3 (an instance of BART finetuned to extract the exhaustive set of triples in a sentence following the Wikidata schema). The utility is the F1 score between the generated and the target set of triples.

Machine Translation (MT) with the WMT14 dataset [22] and a pretrained mBART50 model [250] to translate *English to French*. The notion of utility is the match between the generated and the target translation, as measured by BLEU-4.

Non-Toxic Text Generation based on the Real Toxicity Prompt (RTP) dataset [83] for prompting a GPT2 model. The notion of utility is whether the generated output contains toxic language

	LM calls	Value calls
Greedy Search	N	–
Beam Search	N × B	–
Stochastic Beams	N × B	–
VGBS	N × B	N × B × K
MCTS	N × S	N × S

Table B.1: Coarse complexity analysis of the decoding algorithms used, in terms of LM calls and Value calls. N is the number of tokens to be generated, B the number of beams, K the number of next tokens considered by the value model per beam in VGBS, S the number of simulations per generated token in MCTS. In all our experiments, B=5, K=20, S=50.

or not. The utility function is an ALBERT model [94] trained on the Jigsaw dataset with an unintended bias to measure the toxicity of a text.

Non-Soluble Protein Generation: We use the SwissProt-EF dataset [13] for prompting a ProtoGPT2 model [68], which is pretrained on sequences of amino acids from protein prompts. The notion of utility is whether the generated protein is *soluble* or not. To measure non-solubility, we use ProtBERT [62], which is a BERT-based model trained on a large corpus of protein sequences in a self-supervised fashion. Finally,

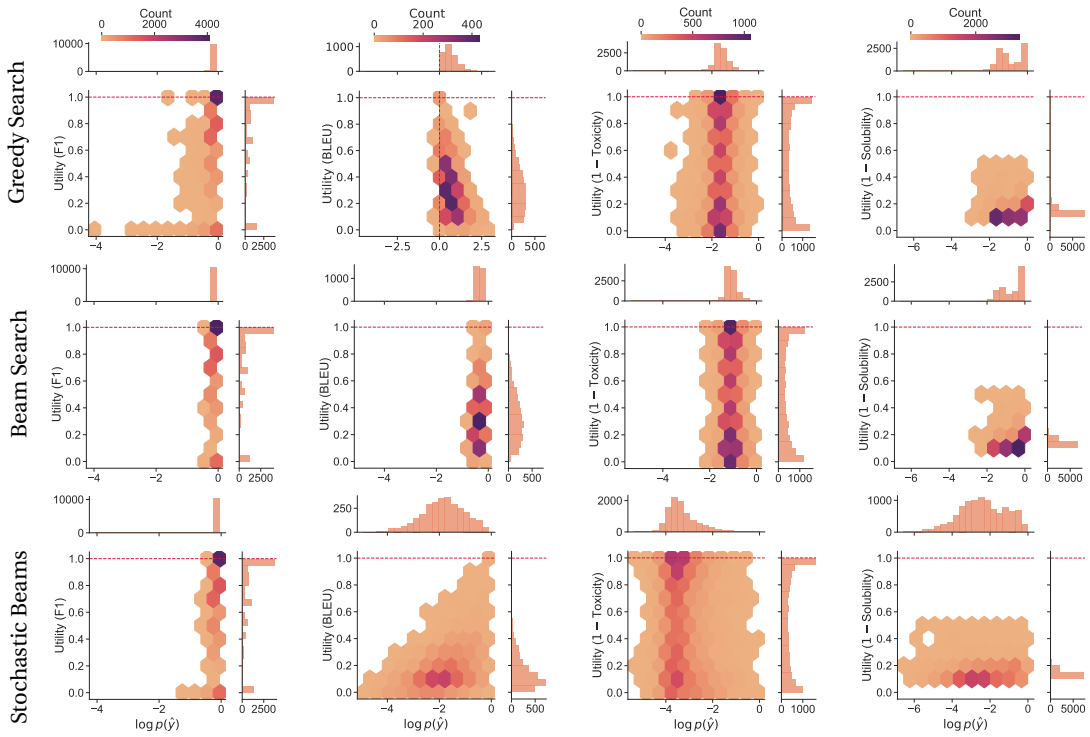
Sports Understanding with the Sports Understanding (SU) task, part of the BIG-bench effort [20], with a 530B parameter pre-trained language model: MT-NLG [236]. The primary purpose of this task is to test the general understanding of sports by asking the model to discriminate between plausible and implausible statements relating to sports.

B.2.2 Hyperparameters of Decoding Algorithms

The number of beams for BS, SB, and VGBS is fixed to 5 for all tasks, except for cIE, where it is 10 — the model's default; and the number of simulations in MCTS is fixed to 50. Due to the high computational cost, to decide the optimal value for MCTS's c_{puct} and VGBS's α in RQ3, we run a hyperparameter search for each level of noise over a small sample of 80 data points (see Appendix B.3.3 for the ranges of the search). For both of the prompting-based strategies, we use greedy decoding during inference.

B.2.3 Complexity Analysis of Decoding Algorithms

Most of the compute during decoding is allocated on querying the LM or the value model. Therefore, to show how decoding strategies compare in terms of the computation cost, in Table B.1 we provide a coarse complexity analysis in terms of the LM and value model calls.



(a) cIE (REBEL) (b) MT (WMT14) (c) NTTG (RTP) (d) NSPG (SwissProt)

Figure B.1: **RQ1 (post-decoding alignment) – without target normalization**: A version of Fig. 4.2 without the target answer log-likelihood normalization.

ID	Input	Prediction
1	Is the following sentence plausible? Leon Draisaitl grounded out to second base in the National League Championship Series. A:	i'm pretty sure that's not really a grammatical sentence, although one might...
2	Is the following sentence plausible? John Collins threw a touchdown in the NFC divisional round. A:	the nor gates give you (not(a)) and (not(b))
3	Is the following sentence plausible? Jack Eichel dunked the ball. A:	hmm? well, i'm not really sure. let me look it up in

Table B.2: Examples of outputs that are not providing an answer. The first and the second row provide an example where the model produces unrelated text, while the third row is an example of an indefinite answer.

B.3 Experiments and Results

B.3.1 RQ1: The Likelihood–Utility Relationship

Fig. B.1 is an alternative version Fig. 4.2 where the x-axis (for cIE and MT) is not normalized using the log-likelihood of the target answers.

B.3.2 RQ3: Prompting as an MMS

Extracting Labels from Zero-Shot Predictions

The outputs produced with zero-shot prompting do not follow a particular structure that can be used to extract the answer and, therefore, need to be processed manually. In some cases, it was not possible for an answer to be extracted. The two most common reasons for this were unrelated text as an answer or an indefinite answer. We provide examples of such predictions in Table B.2. Overall, 24.9% of the answers could not be parsed. In such cases, we favored putting an indefinite label instead of "yes" or "no", and counting the answer as wrong irrespective of the ground truth label. If the answer and explanation were unrelated, but an answer was given, we did consider the answer.

B.3.3 Computational Infrastructure and Runtime

	Number of Beams	Time (in GPU hours)
cIE + Greedy	1	1.5
cIE + BS	10	10.5
cIE + SB	10	10.5
MT + Greedy	1	0.5
MT + BS	5	1.0
MT + SB	5	1.5
NTTG + Greedy	1	2.0
NTTG + BS	5	3.5
NTTG + SB	5	4.5
NSPG + Greedy	1	3.5
NSPG + BS	5	5.5
NSPG + SB	5	7.0

Table B.3: **Parameters for the greedy likelihood-based decoding algorithms.** The default parameters for each model were used, and no hyperparameter search was conducted.

The evaluation for RQ1 as well as the hyperparameter search for RQ2 were conducted on a single machine with 24 Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz processor cores and 441 GB of RAM, equipped with 4 NVIDIA V100-PCIE-16GB GPUs. Table B.3 provides the details for RQ1.

For VGBS, we performed a small hyperparameter search over different values for $\alpha = 0.01, 0.25, 0.5, 0.75, 0.99$ on 80 datapoints for each noise value of the value model. The same procedure was conducted for MCTS, over $c_{puct} = 0.25, 1.25, 3$. Each of these runs took 20 to 30 minutes of wall time, that is, slightly over 1 to 2 hours of GPU time. Table B.4 and Table B.5 provide the final parameters for VGBS and MCTS respectively.

The evaluation for RQ2 and RQ3, as well as the hyperparameter search for RQ2, were conducted on a single machine with 96 processor cores and 840 GB of RAM, equipped with 8 NVIDIA A100-SXM4-80GB GPUs. Table B.4 and Table B.5 provide the details for RQ2.

The evaluation for RQ3 was performed following the Sports Understanding setup in Wei et al. [266], by taking the same in-context examples. We used greedy decoding for all of the prompting methods. The running time for all prompting experiments (ZS, FS, and CoT) was 6 GPU hours.

	α	Time (in GPU hours)
MT ($\lambda = 0.99$)	0.01	4
MT ($\lambda = 0.5$)	0.01	4
MT ($\lambda = 0.35$)	0.25	4
MT ($\lambda = 0.25$)	0.25	4
MT ($\lambda = 0.15$)	0.25	4
MT ($\lambda = 0.01$)	0.75	4
NTTG (oracle)	0.25	32
NTTG (1200 steps)	0.25	30
NTTG (400 steps)	0.25	30
NTTG (200 steps)	0.25	29

Table B.4: **Parameters for VGBS.** For all the experiments, the value models consider the top-10 tokens according to the likelihood. The BLEU to the true target is weighted by λ (i.e., high λ translates to high-quality value model).

	c_{puct}	Time (in GPU hours)
MT ($\lambda = 0.99$)	1.25	41
MT ($\lambda = 0.5$)	0.5	41
MT ($\lambda = 0.35$)	0.5	40.5
MT ($\lambda = 0.25$)	0.25	40
MT ($\lambda = 0.15$)	0.25	40
MT ($\lambda = 0.01$)	1.25	40
NTTG (oracle)	1	125
NTTG (1200 steps)	1	96
NTTG (400 steps)	1	100
NTTG (200 steps)	1	98

Table B.5: **Parameters for MCTS.** For all the experiments, at each node, we consider the top-20 tokens according to the likelihood and perform 50 simulations. The BLEU to the true target is weighted by λ (i.e., high λ translates to high-quality value model).

C Appendix for Chapter 5

C.1 LLM cIE failure cases

In Fig. C.1 we showcase examples of LLMs not being able to solve the problem of cIE effectively. While `text-davinci-003` can often identify the core information in the text, it is not able to map the names to the Wikidata entities or relations.

C.2 Synthetic Data Generation

In this section, we give details on the sampling and the synthetic data generation process. Additionally, we provide examples of prompts used to generate the data in Fig. C.2, as well as generated sentences with `text-davinci-003` and `code-davinci-002` in Table C.2.

C.2.1 Details about the Knowledge Graph

We first select entities and relations which appear in the REBEL dataset as described in Sec. 5.3.1 and relations that do not take literal arguments. We filter out all the entities whose names cannot be associated with a Wikipedia page. This subset comprises 2,715,483 entities and 888 relations. We also exclude entities with a degree of 0, as they do not contribute to any facts in the graph. As a result, our filtered graph includes 2,715,483 nodes and 17,655,864 edges. It is worth noting that our synthetic data generation approach can easily be applied to larger subsets of the Wikidata KG or even the full KG. However, we subsample the KG to remain comparable to previous research.

C.2.2 Triplet Sampling

To sample a coherent triplet set, we follow the iterative procedure explained in Sec. 5.3.2. The parameters involved in this procedure: (i) the *number of triplets* per triplet sets and (ii) the *bias factor*. We sample the number of triplets from a Poisson distribution of mean 3. Assuming

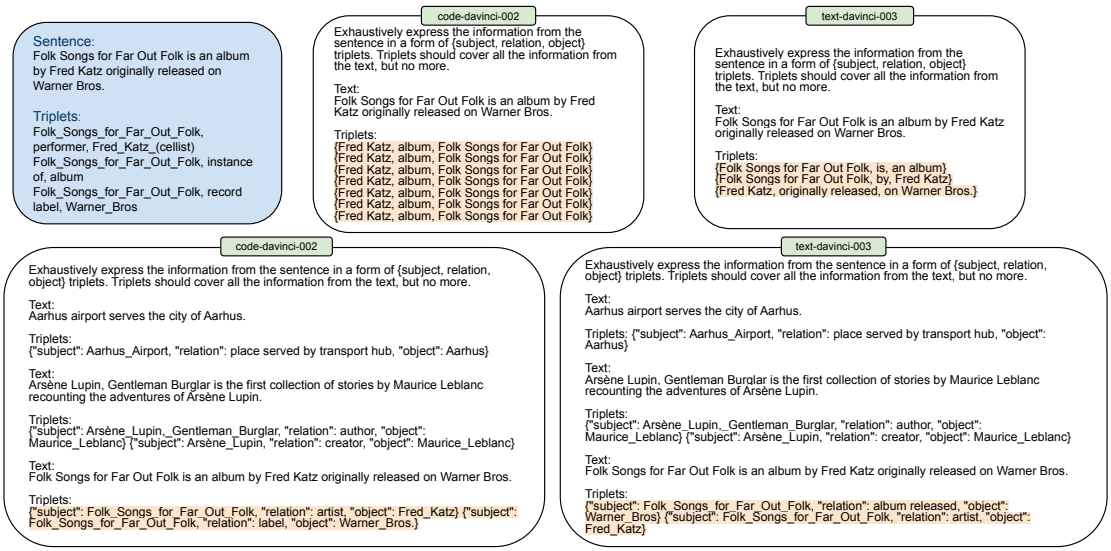


Figure C.1: **Examples of failure cases of LLMs attempts to solve cIE task.** In some cases, models are not able to recognize all the facts present in the sentence. Even when this is possible, they are not able to map subjects, relations, and objects to Wikidata concepts.

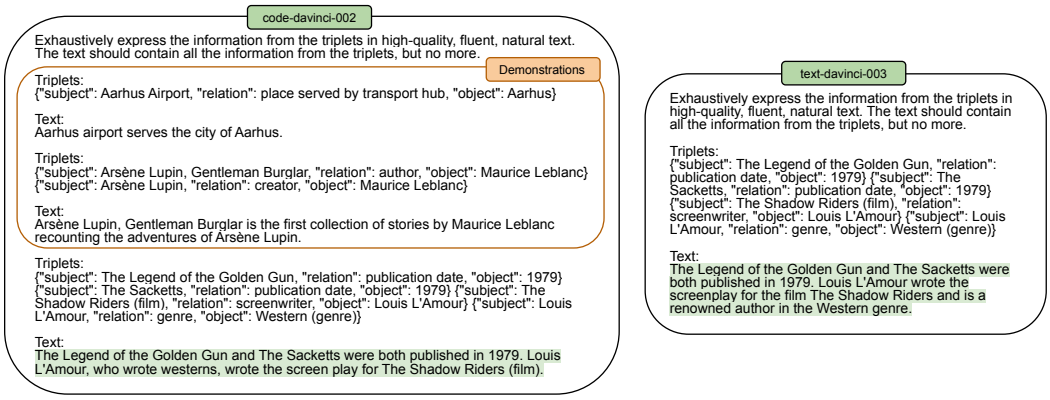


Figure C.2: **Best performing prompts.** We present the best-performing prompts for both models, `text-davinci-003` and `code-davinci-002`. `code-davinci-002` makes use of demonstrations. Text highlighted in green corresponds to the output of the model.

a triplet set comprising N different entities, the next (subject or object) entity is sampled from a probability distribution where entities: (i) not appearing in the triplet set are sampled with probability proportional to 1; (ii) entities already appearing in the triplet set are assigned a probability proportional to $(N+1-r)^{bf}$, where bf is the bias factor, and r is the rank of the current triplet. To choose the bias factor, we fix a random seed for starting nodes and apply the sampling procedure described above for varying choices of bias factor ([1, 3, 7, 10]). We manually inspect the result triplet set and judge their coherence. We find that 7 and 10 yield good coherence but 7 allows for more diversity, whereas 10 always focuses on a single anchor entity. Therefore, we fix the bias factor to 7.

parameter	code-davinci-002	text-davinci-003
max_tokens	100	50
temperature	0.7	0.7
top-p	1	1
frequency_penalty	0.2	0.2
presence_penalty	0	0
stop	"\n"	"\n"
n	1	1
best_of	5	1

Table C.1: **Optimal generation parameters for the LLMs used in the SDG.**

For sampling the starting point, we opt for mixed strategy described in Sec. 5.3.2. The parameters for this procedure are the dampening factor d of the entity and relation distributions as well as K: how often do we recompute the entity and relation distribution from the empirical observation in the current sample (and switch strategy). To choose these parameters, we sample 120K data points with varying d ([0.01, 0.05, 0.1, 0.5, 1]) and K ([2K, 10K, 20K, 120K]). A dampening factor of 1 means no dampening, and $K = N$ the number of samples means no computation of empirical distributions and only using the relation-based strategy. We then look at the skewness, entropy and median number of appearance of relations among the 120K sampled triplet sets, as well as the number of entities covered. We found that a good compromise is given by $d = 0.01$, $K = 20K$.

C.2.3 Triplet Set to Text

Choice of LLM. The `code-davinci-002` model has been trained on a mix of language and code with further instruction finetuning. `text-davinci-003` was further finetuned with reinforcement learning from human feedback making it more effective at zero-shot learning with instructions but less capable of in-context learning. We query the models through the OpenAI API.

Inference costs. At the time of writing, `code-davinci-002` was free with a limitation of 20 requests and 150k tokens per minute. Therefore, the cost for constructing the Wiki-cIE Code dataset was $ 0. `text-davinci-003` was priced at $ 0.02 per 1K tokens, and the total cost of constructing the Wiki-cIE Text was $ 223.55.

C.2.4 Distributional Properties of the Data

In Fig. C.3, we showcase the cumulative distribution function of the relation frequencies in REBEL, Wiki-cIE Code and Wiki-cIE Text, providing a more complete view of how REBEL differs from the synthetically generated datasets on this important dimension.

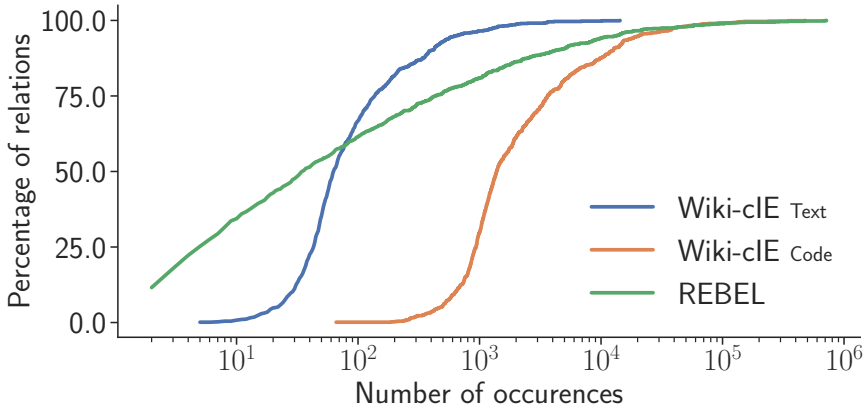


Figure C.3: **Cumulative distribution function (CDF) plot of the relation frequencies in each dataset.** The relation frequencies in Wiki-cIE Code and Wiki-cIE Text have a similar CDF graph — follow a similar distribution — shifted on the x-axis due to the difference in the dataset size. In contrast, the relation frequency distribution in REBEL is heavily skewed, with most relations having few occurrences. Despite being larger than Wiki-cIE Code, more than half of the relations in REBEL have fewer occurrences than the least frequent relation in Wiki-cIE Code.

C.2.5 SDG Evaluation

B.5.1 Computing Performance Metrics

To compute the precision and recall (*cf.* Appendix A.3), we need the number of: (i) target triplets; (ii) correctly predicted triplets; (iii) triplets expressed in the text (*i.e.*, the total number of predicted triplets).

Number of target triplets. This number can be trivially calculated by simply counting the number of triplets in the target set, which comes from REBEL.

Number of correctly predicted triplets. We estimate this quantity by hiring Amazon Mechanical Turk (MTurk) workers to annotate the data. They are presented with a sentence and a set of triplets and asked to select those actually expressed in the sentence. A more detailed description of the task is provided in Appendix C.2.5.2. Appendix C.2.5.3 details the quality checks and the inter-annotator agreement for the task. In conclusion, this procedure leaves us with the number of correctly predicted triplets and allows us to compute the micro-and macro-recall.

Number of triplets expressed in the text. For this analysis, we aim to exhaustively annotate all the information in the text, even that which is not contained in the triplet set. However, doing this properly requires a non-trivial understanding of the closed information extraction task. Because of that, we opted to estimate this number with three of the authors acting as annotators. Appendix C.2.5.4 contains the details of the annotation procedure. Given the

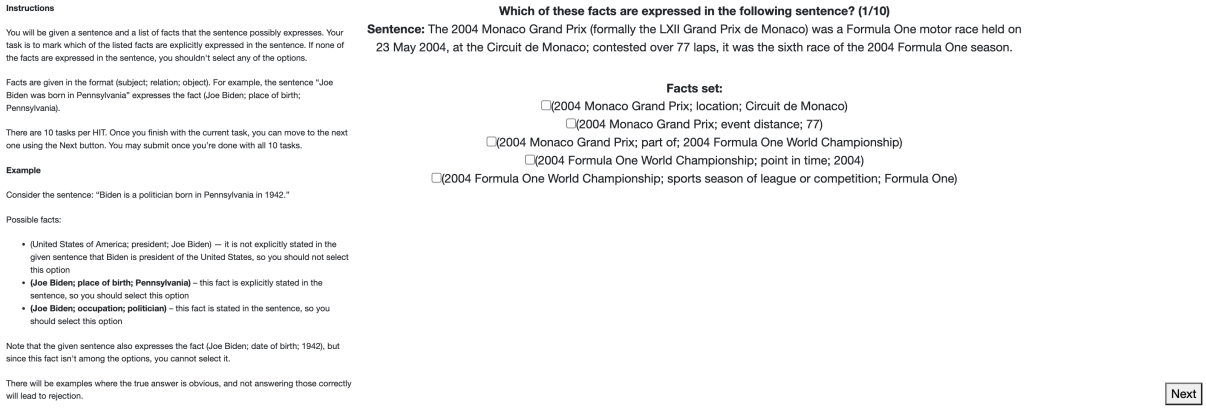


Figure C.4: **Mturk setting.** Workers are presented with the sentence and a list of triplets. Their task is to decide which triplets are present in the presented sentence. They are also presented with detailed instructions and examples.

number of total triplets in the text, we can calculate the micro-precision using the standard definition. This estimate of the precision is, in fact, a (very) tight upper bound on the true precision for each model, as there may exist triplets that were accidentally missed by the annotators. Finally, we note that to enable exhaustive annotation, we relaxed the catalog constraints on the relations in the annotated triplets, which prevented us from computing the macro-precision.

B.5.2 Human Annotation Task

For each annotation task, we presented the MTurk workers with a single text. Along with the text, a list of potential triplets was provided, and the workers were instructed to indicate which of the triplets were present in the text. In the instructions, we included an example text, a set of triplets paired with explanations why each should or should not be marked, as depicted in Fig. C.4.

B.5.3 Quality of annotations

To ensure high-quality annotations, we took the following measures:

- **Crowdworkers criteria.** In an attempt to get reliable ratings, we recruited workers that have previously completed at least 1000 tasks with 99% acceptance rate. Workers were restricted by location to US, UK and Canada. We targeted a pay rate of $ 8-10 per hour, guided by US minimum wage.
- **Honeypots.** Each MTurk task consisted of 10 rating tasks. Among the 10 rating tasks, there were 1-2 honeypots, each constructed by randomly selecting two different REBEL samples and pairing the sentence from the first sample with the target set from the

second. In this case, workers were expected not to mark any of the given triplets, as none of them were expressed in the sentence. This allowed us to filter out unreliable workers.

- **Multiple annotators.** Each task was done by three different workers. The final set of triplets expressed in the sentence was constructed by considering the majority vote for each of the triplet in the set. The Fleiss' kappa was 0.2239.

B.5.4 Precision Annotation Procedure

The annotation was performed in two steps. In the first step, one of the annotators annotated every sentence with the triplets expressed in the text external to the original triplet set. In the second step, two other annotators independently decided whether each annotated triplet was indeed expressed in the generated text. Conflicts were resolved via discussion. This was done without knowing whether the text-triplets pair came from REBEL or one of our synthetically generated datasets. The estimated prediction set was constructed as the union of the original triplet set and the missing triplets that the annotators provided. 6 data points were excluded from the final analysis due to problems with the original data.

C.3 Experiment Implementation Details

Data. The train, test, and validation splits are inherited for REBEL [111] and sampled at random for the newly released datasets. We restrict the data points to those with input and target sequences with at most 256 tokens. To facilitate reproducibility, we release the exact splits used in our experiments.

Training. The models were trained using the Adam optimizer with a learning rate of 3e-4, 0.1 gradient clipping on the Euclidean norm, and a weight decay of 0.05. We trained the models for 8000 steps, with a batch size of 2568, and a polynomial learning rate scheduler with 1000 warm-up steps and a final learning rate of 3e-05. Following the hyper-parameter optimization for GenIE (Chapter 3), we used the default values for most of the parameters and tuned: the number of training and warm-up steps, the batch size, and the weight decay. Importantly, due to the high training costs, the parameters were tuned on the REBEL dataset (the GenIE models), and the best-performing set of parameters was reused for training on the synthetic datasets (the SynthIE models). Having a better-optimized set of hyper-parameters set gives an advantage to the baseline models, which we expect to be insignificant. The complete parameter configuration and the code to reproduce the experiments is available in the GitHub repository provided in the abstract.

Inference. Following the evaluation setup in Chapter 3, we use Constrained Beam Search with 10 beams. We normalize the log probabilities by sequence length and allow for any number of n-gram repetitions. Additionally, we experimented on the validation set with the

triplets	rebel text	text-davinci-003	code-davinci-002
(Samsung Galaxy Xcover 3; operating system; Android (operating system)), (Samsung Galaxy Xcover 3; manufacturer; Samsung Electronics)	The Samsung Galaxy Xcover 3 is an Android smartphone produced by Samsung Electronics and released in July 2015.	**The Samsung Galaxy Xcover 3 is a smartphone manufactured by Samsung Electronics that runs on the Android operating system.**	**The Samsung Galaxy Xcover 3 is a smartphone manufactured by Samsung Electronics that runs the Android operating system.**
(Cleveland Cavaliers; league; National Basketball Association)	During his freshman year he helped lead them to a West Coast Athletic Conference championship and a berth into the 1985 NCAA Tournament, where Pepperdine would lose in the first round. Middlebrooks was named the WCAC Freshman of the Year. The following season, the Waves once again won the WCAC, qualified for the 1986 NCAA Tournament, and lost in the first round. This would be Middlebrooks' last experience with the NCAA Tournament as the Waves did not make it during his junior season and only qualified for the 1988 National Invitation Tournament in his senior year.	**The Cleveland Cavaliers are a professional basketball team that plays in the National Basketball Association (NBA).**	The Cleveland Cavaliers are a professional basketball team based in Cleveland, Ohio. They compete in the National Basketball Association (NBA) as a member of the league's Eastern Conference Central Division.
(Pix Brook; located in the administrative territorial entity; Hertfordshire), (Pix Brook; mouth of the watercourse; River Hiz), (Hertfordshire; shares border with; Bedfordshire), (Bedfordshire; shares border with; Hertfordshire), (River Hiz; tributary; Pix Brook)	Pix Brook is a stream that flows through Letchworth Garden City in Hertfordshire, Stotfold in Bedfordshire, and meets the River Hiz north of Arlesey.	Pix Brook is located in the administrative territorial entity of Hertfordshire, and is the mouth of the watercourse River Hiz, which is a tributary of Pix Brook. Hertfordshire shares a border with Bedfordshire, and likewise Bedfordshire shares a border with Hertfordshire.	**Pix Brook is a tributary of the River Hiz and is located in Hertfordshire, which borders with Bedfordshire.**
(1999 Nunavut general election; point in time; 1999)	David Iqaqrialu (born July 2, 1954) is a former territorial level politician from Clyde River, Northwest Territories (now Nunavut), Canada. He served as a member of the Nunavut Legislature from 1999 until 2004.	**The 1999 Nunavut general election took place in 1999.**	**The 1999 Nunavut general election was held in 1999.**
(Poltava Governorate; country; Russian Empire)	Andriy Bobyr was born in the village of Nychyporivka, in the Poltava Governorate of the Russian Empire.	**The Poltava Governorate was part of the Russian Empire.**	**Poltava Governorate was part of the Russian Empire.**
(United States; contains administrative territorial entity; Wisconsin), (Wisconsin; located in the administrative territorial entity; United States)	This genus contains six species that are distributed across the Eastern continental United States as far west as Oklahoma and Wisconsin.	The United States contains the administrative territorial entity of Wisconsin, which is located within the United States.	**Wisconsin is a state in the United States.**
(Two Weeks with the Queen; publication date; 1990), (Two Weeks with the Queen; author; Morris Gleitzman)	Two Weeks with the Queen is a 1990 novel by Australian author Morris Gleitzman.	**Two Weeks with the Queen, a novel by Morris Gleitzman, was first published in 1990.**	**Morris Gleitzman's book "Two Weeks with the Queen" was published in 1990.**
(Ciudad del Este; country; Paraguay)	Antonio Oddone Sarubbi", is a football stadium in the city of Ciudad del Este, Paraguay.	**Ciudad del Este is located in Paraguay.**	**Ciudad del Este is a city in Paraguay.**

Table C.2: **Text comparison for REBEL triplet sets** This table contains the original REBEL data, as well as text generated using two OpenAI models, `code-davinci-002` and `text-davinci-003`, for the same triplet sets. Synthetic data samples are better in terms of recall, especially precision, and remain fluent. Samples that are overall better in terms of precision, recall, and fluency are bolded.

Dataset	*# Data Points*			*# Triplets*			*# Entities*			*# Relations*		
	train	val	test	train	val	test	train	val	test	train	val	test
REBEL	2,813,210	155,926	156,449	7,187,915	397,326	398,252	2,038,741	205,080	205,549	1071	691	690
REBEL⋆	2,069,780	114,448	114,953	4,642,624	256,327	257,129	1,537,472	151,617	151,997	865	595	580
Wiki-cIE Code	1,815,378	10,000	50,286	6,055,911	34,262	172,991	1,806,126	27,553	105,176	888	883	888
Wiki-cIE Code⋆	1,669,708	9,222	46,210	5,482,658	31,073	156,350	1,668,198	25,295	96,337	876	869	876
Wiki-cIE Text	–	10,000	50,286	–	34,262	172,991	–	27,553	105,176	–	883	888
Wiki-cIE Text⋆	–	9,230	46,295	–	31,117	156,805	–	25,323	96,483	–	869	876

Table C.3: **Statistics of the datasets.** ⋆The filtered version of the dataset used in this work. We filter out data points that have: (i) triplets in their respective target set corresponding to entities and relations outside of the pre-defined knowledge base constrained (*cf.* Sec. 5.4.1); (ii) input longer than 256 tokens; (iii) linearized output longer than 256 tokens (always according to the longer fully expanded linearization schema in order to keep the same data points across runs).

value of the length penalty parameter. The value of 0.8 was optimal for the models with fully-expanded linearization and 0.6 for the models with subject-collapsed linearization. The other parameters are kept to their default values.

Infrastructure and compute time. For training all of the models except for the SynthIE T5-large, we used a single machine with 24 Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz processor cores and 441 GB of RAM, equipped with 4 Tesla V100-PCIE-16GB GPUs. Each training run took 40-45 wall-clock hours (160-180 GPU hours), and each inference run on REBEL took around 16 wall-clock hours (64 GPU hours) and 6 wall-clock hours (24 GPU hours) on the Wiki-cIE Code and Wiki-cIE Text datasets.

The SynthIE T5-large model was trained on a machine with 96 Intel(R) Xeon(R) CPU @ 2.20GHz processor cores and 680 GB of RAM, equipped with 8 Tesla A100-PCIE-40GB GPUs. A training run, in this case, took around 30 wall-clock hours (240 GPU hours), and an inference run took around 11 and 4 wall-clock hours (88 and 24 GPU hours) on REBEL, and the Wiki-cIE Code or Wiki-cIE Text datasets, respectively.

C.4 Output Linearization

In Sec. 5.4.1, we introduced two output linearization methods: (i) *fully expanded* (FE) and (ii) *subject collapsed* (SC). This section expands on the two methods. Table C.4 presents example outputs.

Fully expanded linearization (FE). FE mapping, used by GenIE, starts by linearizing each (subject, relation, object) triplet by using the delimiters [s], [r], [o] to demarcate the start of the subject entity, the relation type, and the object entity, respectively. The end of each triplet is demarcated with [e]. The final representation y is constructed by concatenating the textual representations of all of the triplets in the set y_{set}. The advantage of this method lies in its simplicity. However, with most textual sequences introducing several facts per entity (most

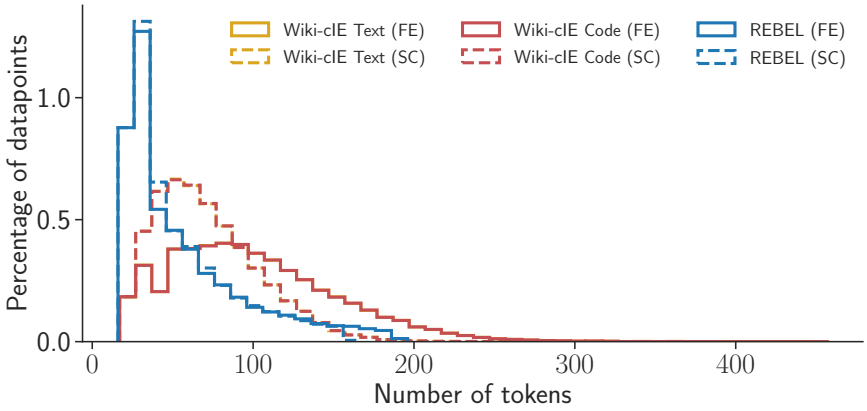


Figure C.5: **Histogram of the number of output tokens according to different linearization strategies.** The subject-collapsed (SC) linearization results in shorter sequences – an effect which is particularly pronounced for Wiki-cIE Code and Wiki-cIE Text where the triplet sets are more exhaustive.

concerning the same subject), many entities are repeated. To alleviate this, we consider the SC mapping.

Subject collapsed linearization (SC). SC mapping, used by Huguet Cabot and Navigli [111], starts by grouping all of the triplets based on the subject and then linearizing them separately by expressing the group's subject once and then listing the relation and object for each of the triplets in alternating fashion, demarcating the start of each part with the previously introduced delimiters. The end of each triplet group is demarcated by the same delimiter [e], and the final representation is constructed by concatenating the textual representations of all of the triplet groups. The SC linearization results in shorter target sequences at the cost of more complex subpart dependencies in the output. Figure Fig. C.5 captures this effect for the datasets considered in this work. This effect is particularly pronounced for Wiki-cIE Code and Wiki-cIE Text where the triplet sets are more exhaustive, leading to repetitions. For REBEL, which is missing many triplets from the target set the difference in output length between the two linearization methods is not substantial.

While the sequence representation has an intrinsic notion of order, the output set of triplets does not. To mitigate the effects of this discrepancy, we enforce a consistent ordering of the target triplets during training by considering first the triplets corresponding to subjects appearing earlier in the sentence. Ties are resolved by the appearance position of the object entity. Whenever the triplets' entities are not linked to entity mentions in the textual input, we use a heuristic that links each entity to the largest sequence of words from the textual input appearing in the entity name (or to the beginning of the sentence in case of no overlap).

Triplet Set	(Mount Lanning; instance; Mountain), (Mount Lanning; mountain; Sentinel Range), (Newcomer Glacier; mountain; Sentinel Range)
Fully Expanded	[s] Mount_Lanning [r] instance of [o] Mountain [e] [s] Mount_Lanning [r] mountain range [o] Sentinel_Range [e] [s] Newcomer_Glacier [r] mountain range [o] Sentinel_Range [e]
Subject Collapsed	[s] Mount_Lanning [r] instance of [o] Mountain [e] [r] mountain range [o] Sentinel_Range [e] [s] Newcomer_Glacier [r] mountain range [o] Sentinel_Range [e]

Table C.4: **Example for the different linearization methods.** In the first row, we showcase the original triplet set. The following two rows show the linearization of the original triplet set according to the two methods we consider: fully expanded and subject collapsed.

C.5 SynthIE in Action

C.5.1 Human Evaluation of REBEL

G.1.1 Constructing REBEL Clean

Choice of data points. We started by randomly selecting 1000 data points from REBEL's test set, ordering them according to their corresponding numeric identifier (ID), and printing their ID and input text (crucially, *without* looking at the target triplets). By manually inspecting them in order, we selected data points until we reached 360.[1] We used two simple criteria for choosing the data points.
Criterion 1: The text should have substantial "extractable" information.
One of the samples that were discarded due to this criterion is "In addition to saxophone, he plays clarinet, bass clarinet, French horn, flute, and cornet.".
Rationale: Without extractable information, we would be spending our "sample budget" on examples that cannot be resolved even in theory and are, therefore, misleading and not informative.
Criterion 2: The central information in the text should not be of a literal type.
One of the samples that were discarded due to this criterion is: "Incumbent Republican Alfred E. Driscoll defeated Democratic nominee Elmer H. Wene with 51.54% of the vote."
Rationale: GenIE (Chapter 3) does not support literals. While extending GenIE (and therefore SynthIE) to literals is possible, that is not the focus of our work. To isolate the effect of synthetic data, we choose to keep the same setting as Chapter 3 and limit our SDG to relations that do not rely on literal arguments. Therefore, following the same reasoning as in Criterion 1, data points for which the central information involves literals would be misleading and not informative.

[1] 350, and 10 as a backup in case of annotation issues, but we did not encounter any issues and kept all of them.

Construction of target-triplet candidate sets. It is relatively straightforward and accurate to establish which of the triplets in a given set are expressed in a specific text by considering whether each of them is expressed separately (see Appendix C.2.5). However, providing workers with our entity catalog containing around 2.6M and the relation catalog of 888 items and asking them to annotate the sentence exhaustively is a difficult task that is bound to result in inaccurate annotations. To circumvent this issue, instead of asking annotators to annotate a sentence exhaustively, we cast the task of "almost" exhaustive annotation in the same format as the task of estimating precision by providing the annotators with a larger "target set" of candidate triplets (with high recall) and ask them to establish which of triplets are indeed expressed in the text. This process will be as exhaustive as the recall of the candidate set, i.e., we will correctly detect the triplets within the candidate set that are expressed in the text and overlook any triplets outside of the candidate set.

In the construction of REBEL Clean, for each datapoint, we construct the candidate target set as the union of the triplets in the corresponding target set in REBEL and the set of predicted triplets by SynthIE T5-large. The rationale here is as follows. We start from the assumption that any candidate target set should be a super-set of REBEL's original target set, which we know is not exhaustive. Therefore, to increase the coverage, we enlarge the target set in REBEL with the predictions of our best model SynthIE T5-large, which is trained to provide exhaustive annotations. The human annotators will then filter out any incorrectly annotated triplets in the candidate target set.

This design of the human-annotation procedure ensures that (i) our estimate of the true precision will be correct (up to human error) and (ii) the measured recall will be an upper bound with a constant multiplicative gap to the true recall for all the models of interest (equal to the portion of triplets that are potentially missing from the curated target set). We discuss these consequences in Sec. C.5.1.2 and Sec. C.5.1.3

Human annotation task. This annotation task followed the same format as the task described in Appendix C.2.5.2. However, to ensure the highest possible quality, for this task, instead of relying on MTurk workers, we hired two Ph.D. and two MSc students, who were not familiar with our work (to avoid potential bias). For their time, they were compensated 25 CHF per hour.

We conducted the annotation in two stages. In the first stage, one of the Ph.D. students annotated all of the data points, while each MSc student annotated half of the data points such that every data point was annotated twice. In the second stage, the second Ph.D. student annotated each data point where the two annotations (from the first Ph.D. student and from one of the MSc students) did not match, resolving the conflicts.

G.1.2 Precision on REBEL Clean

To compute the precision (see Appendix A.3), we need the number of: (i) predicted triplets; and (ii) correctly predicted triplets.

The total number of predicted triplets across all documents $d \in \mathscr{D}$, by the model m, is given as:

$$P_m = \sum_{d \in \mathscr{D}} |P_{d,m}| \tag{C.1}$$

where $P_{m,d}$ is the set of triplets predicted by model m for document d. This number can be trivially calculated for any model and dataset.

Let G_d be the gold set of triplets in the REBEL Clean dataset and $G_d^* = G_d \cup U$ the *true* set of target triplets where U corresponds to the set of triplets in the text that are missing from G_d. With this notation, the total number of correctly predicted triplets across all documents $d \in \mathscr{D}$, by the model m, is defined as:

$$C_m^* = \sum_{d \in \mathscr{D}} |P_{m,d} \cap G_d^*| \tag{C.2}$$

Since G_d^* is unknown in practice, to estimate precision, we rely on

$$C_m = \sum_{d \in \mathscr{D}} |P_{d,m} \cap G_d| \,. \tag{C.3}$$

However, in the construction of the gold set of triplets in REBEL Clean, for document d, the human annotators considered the triplets $A_d = P_{d,REBEL_{Gold}} \cup P_{d,SynthIE_{T5-large}}$ where $P_{REBEL_{Gold}}$ corresponds to the gold (target) set of triplets in REBEL (as the predictions of a model REBEL$_{Gold}$) and $P_{SynthIE_{T5-large}}$ to the predictions from SynthIE T5-large. As a consequence, for REBEL$_{Gold}$ and SynthIE T5-large the estimated and the true total number of correctly predicted triplets, and therefore the estimated and the true micro- and macro-precision will be equivalent (up to human error). More generally, for any model m, if:

$$\sum_{d \in \mathscr{D}} |P_{m,d} \cap G_d^*| \approx \sum_{d \in \mathscr{D}} |P_{m,d} \cap A_d \cap G_d^*| \tag{C.4}$$

the true precision will be estimated well by the standard precision on the REBEL Clean. Considering that GenIE was trained on REBEL, and SynthIE T5-base is a smaller version of SynthIE T5-large, this criterion should be satisfied, and therefore, we expect to have good estimates of micro- and macro-precision for these models as well.

G.1.3 Recall on REBEL Clean

The previous section argues that, for all of the models of interest, $C_m \approx C_m^*$. As a consequence, using the same notation as the previous section, for the computation of the micro-recall (see Appendix A.3) of model m, we have:

$$\begin{aligned}
\text{micro-R}_m^* &= \frac{C_m^*}{\sum_{d\in\mathscr{D}} |G_d^*|} \\
&\approx \frac{C_m}{\sum_{d\in\mathscr{D}} |G_d^*|} \\
&= \frac{C_m}{\sum_{d\in\mathscr{D}} |G_d|} \times \frac{\sum_{d\in\mathscr{D}} |G_d|}{\sum_{d\in\mathscr{D}} |G_d^*|} \\
&= \text{micro-R}_m \times \text{micro-R}_{REBEL\,Clean}^*
\end{aligned}$$

A similar argument can be made for the macro-recall (see Appendix A.3 for the formal definition). Let $C_m^{(r)}$, $C_m^{*(r)}$, $G_d^{(r)}$, $G^{*(r)}$ denote their respective quantities as before but computed on the subset of triplets corresponding to relation $r \in \mathscr{R}$. Then, for model m, we have:

$$\begin{aligned}
\text{macro-R}_m^* &= \frac{1}{|\mathscr{R}|} \sum_{r\in\mathscr{R}} \frac{C_m^{*(r)}}{\sum_{d\in\mathscr{D}} |G_d^{*(r)}|} \\
&\approx \frac{1}{|\mathscr{R}|} \sum_{r\in\mathscr{R}} \frac{C_m^{(r)}}{\sum_{d\in\mathscr{D}} |G_d^{*(r)}|} \\
&= \frac{1}{|\mathscr{R}|} \sum_{r\in\mathscr{R}} \frac{C_m^{(r)}}{\sum_{d\in\mathscr{D}} |G_d|} \times \frac{\sum_{d\in\mathscr{D}} |G_d^{(r)}|}{\sum_{d\in\mathscr{D}} |G_d^{*(r)}|} \\
&\approx \frac{1}{|\mathscr{R}|} \sum_{r\in\mathscr{R}} \frac{C_m^{(r)}}{\sum_{d\in\mathscr{D}} |G_d|} \times \frac{\sum_{d\in\mathscr{D}} |G_d|}{\sum_{d\in\mathscr{D}} |G_d^*|} \\
&= \text{macro-R}_m \times \text{micro-R}_{REBEL\,Clean}^* \,.
\end{aligned}$$

However, the equivalence, in this case, relies on an additional assumption used in the penultimate equality: the recall of the ground truth annotations in REBEL Clean should be approximately the same across all relations.

D Appendix for Chapter 6

D.1 Data

Example Codeforces and LeetCode problems are provided in Fig. D.1.

In the first experiment, the temporal analysis, we use 239 Codeforces problems ranging from October 2020 to April 2023. In the second experiment, we have 136 problems for Codeforces (some problems are dropped in order to keep the pre-cutoff and post-cutoff buckets equal to 68) and 558 problems for LeetCode (93 for each of the six buckets). Additionally, to support research in the area, we set up an AI competitive coding challenge based on a dataset of Codeforces problems of various difficulties published after the knowledge cutoff date. More details about the CC competition are available in Appendix D.5.

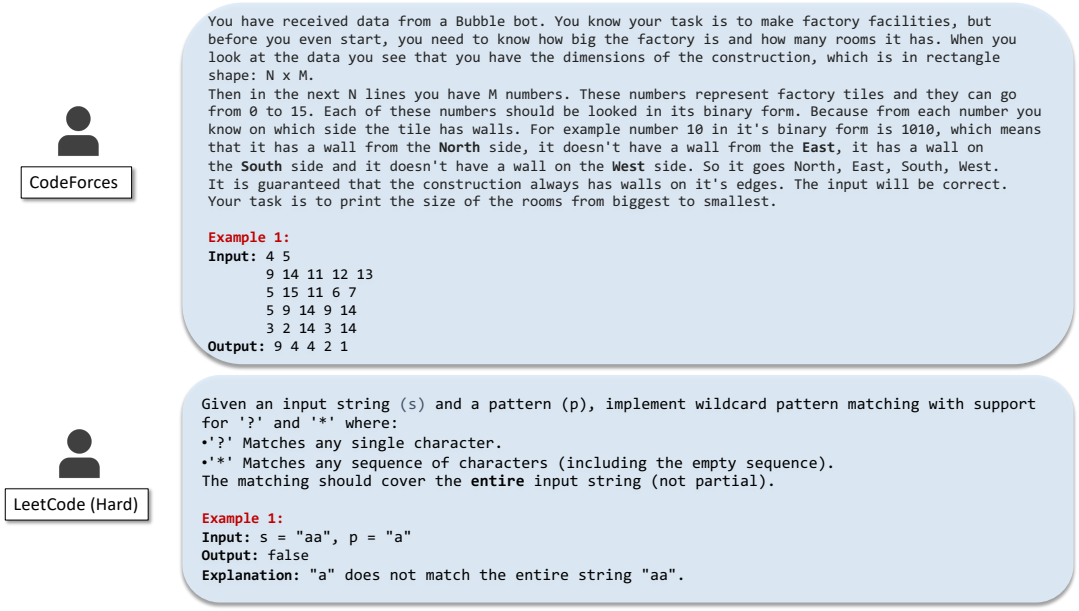


Figure D.1: **Examples of competitive coding problems from Codeforces and LeetCode.**

D.2 Code Testing and Solution Evaluation

The solution evaluation requires a set of input-output pairs, hidden from the user, that comprehensively test the behavior of the program. To compute the final results, we have implemented an online evaluation infrastructure that submits the candidate solutions to the websites' online judges and automatically scrapes the judgment. This mechanism ensures authoritative results.

For many of the Codeforces problems, we managed to scrape (sometimes a subset) of the hidden tests, allowing us to use a faster, local infrastructure for evaluating candidate solutions. On the other hand, LeetCode does not expose any of the hidden tests publicly.

For code testing at inference time, just like a human would, we rely on tests constructed from the (public) input–output example pairs contained in the problem description.

D.3 Concurrent and Previous Works as Specific Instances of Flows

The introduction of LLMs such as BARD, GPT-3, ChatGPT, and its latest version, GPT-4, has led to a breakthrough in AI. This has enabled many exciting developments like CoT, HuggingGPT, AutoGPT, AgentGPT, and BabyAGI. In this section, we demonstrate how *Flows* provides a unified view encompassing concurrent and previous work as specific Flow instances. The details are provided in Figure D.2 and Table. D.1.

1. **Few shot Prompting** (FS) [24] consists in providing a few input-output examples within the prompt, acting as demonstrations to enable the LLM to perform a specific task. This technique relies on the LLM's emergent in-context learning ability to extrapolate from these limited examples and infer how to solve the task in general.

2. **Chain of Thoughts** (CoT) [268] is a prompting method (atomic Flow) that allows LLMs to generate a series of intermediate natural language reasoning steps that lead to the final output.

3. **Tree of Thoughts** (ToT) [288] is a framework that enables (*orchestration*) exploration over coherent units of text (thoughts) that serve as intermediate steps toward problem-solving. ToT allows LLMs to perform deliberate decision-making by considering multiple different reasoning paths and self-evaluating choices to decide the next course of action, as well as looking ahead or backtracking when necessary to make global choices.

4. **Program of Thoughts** (PoT) [38] is a prompting method that allows language models (mainly Codex) to express the reasoning process as a program. The computation is relegated to an external program, which executes the generated programs to derive the answer.

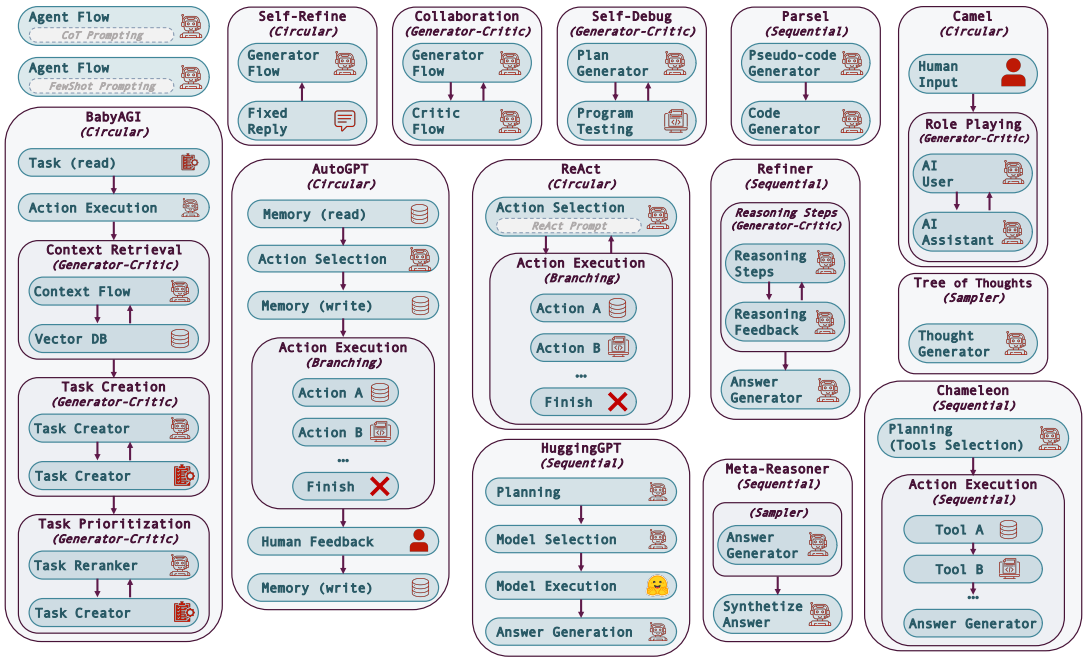


Figure D.2: **Previous works are specific Flows.** We depict a selected subset of previous works incorporating structured reasoning and/or interactions between AI agents, tools, and humans, through the lens of the Flows framework. This demonstrates that Flows is a powerful language for describing, conceptualizing, and disseminating structured interaction patterns.

5. **Mutimodal CoT** (M-CoT) [298] is a method that incorporates language (text) and vision (images) modalities into a two-stage framework that separates rationale generation and answer inference. To facilitate the interaction between modalities in M-CoT, smaller language models (LMs) are fine-tuned by fusing multimodal features.

6. **ToolFormer** [220] is a model that is trained to decide which APIs to call, when to call them, what arguments to pass, and how to incorporate the results into future tokens prediction.

7. **ReAct** [286] is a framework that uses LLMs to generate reasoning traces and task-specific actions sequentially. The framework allows for greater synergy between the two: reasoning traces help the model induce, track, and update action plans and handle exceptions, while actions allow it to interface with external sources, such as knowledge bases or environments, to gather additional information.

8. **Parsel** [295] is a framework that enables the automatic implementation and validation of complex algorithms with code LLMs. The framework first synthesizes an intermediate representation based on the Parsel language and can then apply a variety of postprocessing tools. Code is generated in a next step.

9. **REFINER** [190] is a framework for LMs to explicitly generate intermediate reasoning steps while interacting with a critic model that provides automated feedback on the reasoning.

10. **Self-Refine** [157] is a framework for LLMs to generate coherent outputs. The main idea is that an LLM will initially generate an output while the same LLM provides feedback for its output and uses it to refine itself iteratively.

11. **Recursively Criticize and Improve** (RCI) [122] showed that a pre-trained large language model (LLM) agent could execute computer tasks guided by natural language using a simple prompting scheme where the agent Recursively Criticizes and Improves its output (RCI). Unlike Self-refine, this method uses two separate LLMs (ChatGPT), one for performing the task and another for criticizing.

12. **Self-Correct** [269] is a framework that decouples a flawed base generator (an LLM) from a separate corrector that learns to iteratively correct imperfect generations. The imperfect base generator can be an off-the-self LLM or a supervised model, and the corrector model is trained.

13. **Self-Debug** [40] is a framework that relies on external tools (SQL application or Python interpreter) to help large language models revise and debug SQL commands or Python code with bugs.

14. **Reflexion** [232] is a framework that provides a free-form reflection on whether a step was executed by LLM correctly or not and potential improvements. Unlike self-refine and self-debug, Reflexion builds a persisting memory of self-reflective experiences, which enables an agent to identify its own errors and self-suggest lessons to learn from its mistakes over time.

15. **Meta-Reasoner** [293] is an approach which prompts large language models to meta-reason over multiple chains of thought rather than aggregating their answers. This approach included two steps: (i) ask LLM to generate multiple reasoning chains, (ii) ask another LLM (meta-reasoner) to reason over the multiple reasoning chains to arrive at the correct answer.

16. **HuggingGPT** [229] is a framework that leverages LLMs (e.g., ChatGPT) to connect various AI models in machine learning communities (e.g., Hugging Face) to solve numerous sophisticated AI tasks in different modalities (such as language, vision, speech) and domains.

17. **Camel** [142] is a communicative agent framework involving inception prompting to guide chat agents toward task completion while maintaining consistency with human intentions.

18. **Chameleon** [154] is a plug-and-play compositional reasoning framework that augments external tools with LLMs in a plug-and-play manner. The core idea is that an LLM-based

Flows	**Flow Type**	**Interactions**				**Reasoning Patterns**		**Feedback**	**Learning**
		Self	Multi-Ag.	Human	Tools	Struct.	Plan		
FS [24]	Atomic	✗	✗	✗	✗	✗	✗	✗	✗
CoT [268]	Atomic	✗	✗	✗	✗	✓	✗	✗	✗
ToT [288]	Circular	✓	✗	✗	✓	✓	✗	✗	✗
PoT [38]	Seq.	✗	✗	✗	✓	✓	✗	✗	✗
M-CoT [298]	Seq.	✗	✗	✗	✗	✓	✗	✗	✓
ToolFormer [268]	Seq.	✗	✗	✗	✓	✓	✗	✗	✓
ReAct [286]	Circular	✗	✗	✗	✓	✓	✗	✗	✗
Parsel [295]	Seq.	✗	✓	✗	✓	✓	✓	✗	✗
REFINER [190]	Gen-Crit	✗	✓	✓	✗	✓	✗	✓	✓
Self-Refine [157]	Gen-Crit	✓	✗	✗	✗	✓	✗	✓	✗
RCI [122]	Gen-Crit	✓	✗	✗	✓	✓	✗	✓	✗
Self-Correct [269]	Gen-Crit	✓	✗	✗	✓	✓	✗	✓	✗
Self-Debug [40]	Gen-Crit	✓	✗	✗	✓	✓	✗	✓	✗
Reflexion [232]	Gen-Crit	✓	✗	✗	✓	✗	✗	✓	✗
Meta-Reasoner [293]	Seq.	✓	✓	✗	✗	✓	✗	✗	✗
HuggingGPT [229]	Seq.	✗	✓	✗	✓	✓	✓	✗	✗
Camel [142]	Circular	✗	✓	✓	✗	✓	✗	✓	✗
Chameleon [154]	Seq.	✗	✓	✗	✓	✓	✓	✗	✗
AutoGPT [209]	Circular	✓	✓	✗	✓	✓	✓	✓	✗
BabyAGI [177]	Circular	✗	✓	✗	✓	✓	✓	✗	✗

Table D.1: **Previous work.** We compare previous work across relevant dimensions.

planner assembles a sequence of tools to execute to generate the final response. The assumption is that this will be less error-prone, easily expandable to new modules, and user-friendly.

19. **AutoGPT** [209] is an experimental open-source application that leverages the capabilities of large language models (LLMs) and Chatbots such as OpenAI's GPT-4 and Chat-GPT to create fully autonomous and customizable AI agents. It has internet access, long-term and short-term memory management.

20. **BabyAGI** [177] is an intelligent agent capable of generating and attempting to execute tasks based on a given objective. BabyAGI operates based on three LLM flows: Task creation flow, Task prioritization flow, and Execution flow.

D.4 Prompting

We provide the prompts used to obtain the results in Section 6.6. Our evaluation is made possible thanks to the modular and compositional nature of *Flows*. Some of the experimental setups are deeply nested, and in cases where Flows build on each other, we avoid repetition. Note that the project's GitHub repository provides the code and data to reproduce all of the experiments in the chapter.

Direct prompting for a solution is shown in Listing D.1. To add reflection, we use a Generator-Critic Flow to combine the code generation with a fixed reply, as shown in Listing D.2. In the collaboration setting, we use Listing D.3 as the generator and Listing D.4 as the critic.

Debugging is incorporated via a testing Flow that adds formatting to the output of a code executor. The formatting templates are shown in Listing D.6. To respond to the debug output, we rely on an adjusted coding Flow D.5. Collaboration in the debugging setting is implemented by introducing a critic that provides feedback grounded in the test results. This Flow is detailed in Listing D.3.

The scenarios explained above also support the addition of a planning Flow. An example of plan generation is shown in Listing D.8.

Listing D.1: Prompts for Code Flow (Codeforces)

```
"prompt templates":
  "system_message": |-
    Your goal is to provide executable Python code that solves a competitive
      programming problem. The code should correctly handle all corner cases in order
      to pass the hidden test cases, which are used to evaluate the correctness of the
       solution.

    The user will specify the problem by providing you with:
      - the problem statement
      - input description
      - output description
      - example test cases
      - (optional) explanation of the test cases

    The user will provide you with a task and an output format that you will strictly
      follow.
  "query_message": |-
    # Problem statement
    {{problem_description}}

    # Input description
    {{input_description}}

    # Output description
    {{output_description}}

    {{io_examples_and_explanation}}


    The input should be read from the standard input and the output should be passed to
       the standard output.
    Return Python code that solves the problem. Reply in the following format:
    ```python
    {{code_placeholder}}
    ```
  "human_message": |-
    {{query}}
```

Listing D.2: Prompts for Fixed-Reply Flow

```
"prompt templates":
```

```
  "fixed_reply": |-
    Consider the problem statement and the last proposed solution. Are you sure that
      the solution is provided in the requested format, and crucially, solves the
      problem?
    If that is not the case, provide the corrected version of the code in the following
       format:
    ```python
    {{python_code}}
    ```
    otherwise, reply:
    "Final answer."
```

Listing D.3: Prompts for Code-Collab Flow (Codeforces)

```
"prompt templates":
  "system_message": |-
    Your goal is to provide executable Python code that solves a competitive
      programming problem. The code should correctly handle all corner cases in order
      to pass the hidden test cases, which are used to evaluate the correctness of the
       solution.

    The user will specify the problem by providing you with:
      - the problem statement
      - input description
      - output description
      - example test cases
      - (optional) explanation of the test cases

    The user will provide you with a task and an output format that you will strictly
      follow.
  "query_message": |-
    # Problem statement
    {{problem_description}}

    # Input description
    {{input_description}}

    # Output description
    {{output_description}}

    {{io_examples_and_explanation}}


    The input should be read from the standard input and the output should be passed to
       the standard output.
    Return Python code that solves the problem. Reply in the following format:
    ```python
    {{code_placeholder}}
    ```
  "human_message": |-
    # Feedback on the last proposed solution
    {{code_feedback}}

    Consider the original problem statement, the last proposed solution and the
      provided feedback. Does the solution need to be updated? If so, provide the
      corrected version of the code in the following format:
    ```python
    {{code_placeholder}}
    ```
```

```
    otherwise, reply:
    "Final answer."
```

Listing D.4: Prompts for Code-Collab-Critic Flow (Codeforces)

```
"prompt templates":
  "system_message": |-
    Your goal is to identify potential issues with a competitive programming solution
      attempt.

    The user will specify the problem by providing you with:
      - the problem statement
      - input description
      - output description
      - example test cases
      - (optional) explanation of the test cases
      - a Python solution attempt

    Crucially, your goal is to correctly identify potential issues with the solution
      attempt, and not to provide the code implementation yourself.
    The user will provide you with a task and an output format that you will strictly
      follow.
  "query_message": |-
    # Problem statement
    {{problem_description}}

    # Input description
    {{input_description}}

    # Output description
    {{output_description}}

    {{io_examples_and_explanation}}

    # Python solution attempt:
    ```python
    {{code}}
    ```


    Consider the problem statement and the solution attempt. Are there any issues with
      the proposed solution or it is correct? Explain your reasoning very concisely,
      and do not provide code.
  "human_message": |-
    {{query}}
```

Listing D.5: Prompts for Code-Debug Flow (Codeforces)

```
"prompt templates":
  "system_message": |-
    Your goal is to provide executable Python code that solves a competitive
      programming problem. The code should correctly handle all corner cases in order
      to pass the hidden test cases, which are used to evaluate the correctness of the
       solution.

    The user will specify the problem by providing you with:
      - the problem statement
      - input description
      - output description
      - example test cases
```

```
      - (optional) explanation of the test cases

    The user will provide you with a task and an output format that you will strictly
      follow.
  "query_message": |-
    # Problem statement
    {{problem_description}}

    # Input description
    {{input_description}}

    # Output description
    {{output_description}}

    {{io_examples_and_explanation}}


    The input should be read from the standard input and the output should be passed to
       the standard output.
    Return Python code that solves the problem. Reply in the following format:
    ```python
    {{code_placeholder}}
    ```
  "human_message": |-
    {{testing_results_summary}}

    Consider the problem statement, the last proposed solution, and its issue. Provide
      a corrected version of the code that solves the original problem and resolves
      the issue, without any explanation, in the following format:
    ```python
    {{code_placeholder}}
    ```
```

Listing D.6: Formatting templates for Code-Testing Flow (Codeforces)

```
"formatting templates":
  "no error template": |-
    ${.issue_title}
    All of the executed tests passed.
  "all tests header": |-
    ${.issue_title}
    The Python code does not solve the problem in the problem description due to
      logical errors. It fails on the following tests.
  "compilation error template": |-
    ${.issue_title}
    The execution resulted in a compilation error.
    ## Compilation error message:
    {{error_message}}
  "timeout error template": |-
    ${.issue_title}
    The execution timed out, the solution is not efficient enough.
  "runtime error template": |-
    ${.issue_title}
    The execution resulted in a runtime error on the following test.
    ## [Failed test] Input
    ```
    {{test_input}}
    ```
    ## [Failed test] Runtime error message
```

```
    {{error_message}}
  "single test error": |-
    ${.issue_title}
    The Python code does not solve the problem in the problem description due to
      logical errors. It fails the following test:
    ## [Failed test] Input
    ```
    {{test_input}}
    ```
    ## [Failed test] Expected output
    ```
    {{expected_output}}
    ```
    ## [Failed test] Generated output
    ```
    {{generated_output}}
    ```
  "test error": |-
    ## [Failed test {{idx}}]
    ### [Failed test {{idx}}] Input
    ```
    {{test_input}}
    ```
    ### [Failed test {{idx}}] Expected output
    ```
    {{expected_output}}
    ```
    ### [Failed test {{idx}}] Generated output
    ```
    {{generated_output}}
    ```
```

Listing D.7: Prompts for Code-Debug-Collab Flow (Codeforces)

```
"prompt templates":
  "system_message": |-
    Your goal is to identify the issues with an incorrect competitive programming
      solution attempt.

    The user will specify the problem by providing you with:
      - the problem statement
      - input description
      - output description
      - example test cases
      - (optional) explanation of the test cases
      - an incorrect Python solution attempt and a description of its issue

    Crucially, your goal is to consider all aspects of the problem and pinpoint the
      issues with the solution attempt, and not to provide the code implementation
      yourself.
    Some aspects to consider: Is the input correctly parsed? Is the output correctly
      formatted? Are the corner cases correctly handled? Is there a logical mistake
      with the algorithm itself?
    Use the code execution results provided in the issue description to guide your
      reasoning/debugging.
  "query_message": |-
    # Problem statement
    {{problem_description}}

    # Input description
```

```
    {{input_description}}

    # Output description
    {{output_description}}

    {{io_examples_and_explanation}}

    # Solution attempt to be fixed
    ```python
    {{code}}
    ```

    {{testing_results_summary}}

    Consider the problem statement, the solution attempt and the issue. Why is the
      solution attempt incorrect? How should it be fixed? Explain your reasoning very
      concisely, and do not provide code.
  "human_message": |-
    {{query}}
```

Listing D.8: Prompts for Plan Flow (Codeforces)

```
"prompt templates":
  "system_message": |-
    Your goal is to provide a high-level conceptual solution that, if implemented, will
       solve a given competitive programming problem.

    The user will specify the problem by providing you with:
      - the problem statement
      - input description
      - output description
      - example test cases
      - (optional) explanation of the test cases

    The proposed algorithm should be computationally efficient, logically correct and
      handle all corner cases.

    The user will provide you with a task and an output format that you will strictly
      follow.
  "query_message": |-
    # Problem statement
    {{problem_description}}

    # Input description
    {{input_description}}

    # Output description
    {{output_description}}

    {{io_examples_and_explanation}}


    Return a high-level conceptual solution that would solve the problem. Be very
      concise, and do not provide code.
    Reply in the following format:
    # Conceptual solution
    {{plan_placeholder}}
  "human_message": |-
    {{query}}
```

D.5 The CC-Flows-competition: A New Form of Competitive Coding

Solving competitive coding challenges is an eminently hard problem. The solve rate of only 27% by directly attempting the problem and 47% by the best-performing code Flow, paired with a reliable automatic evaluation metric, makes competitive programming an ideal benchmark for AI systems. Motivated by this, we propose a competition in which, instead of people, Flows solve competitive programming problems.

The competition would leverage the comprehensive dataset of publicly available Codeforces problems and the open-source infrastructure for inference and testing we used in the experiments, available at https://github.com/epfl-dlab/cc_flows. The competition would only include problems published after the knowledge-cutoff date of GPT-4. Furthermore, to avoid overloading the Codeforces online evaluation infrastructure, we filter this dataset to problems for which public and private tests are available, and the output format is compatible with our local code testing infrastructure. Codeforces ranks the difficulty of each problem from 800 to 2100, and we have the following number of problems per difficulty (total of 416):

- difficulty 800: 149
- difficulty 900 to 1500 (inclusive): 185
- difficulty 1600 to 2100 (inclusive): 82

The data should be used following Codeforces' Terms and Conditions. Codeforces prohibits the material from being sold, sublicensed, or commercialized. For more details, see the project's GitHub page.

E Appendix for Chapter 7

E.1 Glossary of Important Terms

Syntactic Token Basic representational units of language, typically defined by the tokenizer in modern NLP systems.

Semantic Token Semantically coherent units, often referred to as *thoughts*, representing meaningful concepts or ideas.

Syntactic Processor Entities that process and manipulate syntactic tokens, such as parsers or language models.

Semantic Processor Entities that process and manipulate semantic tokens, including LLM-based systems, humans, and various tools such as search engines and code executors. Notably, a semantic processor can be implemented by a semantic decoding algorithm itself, showcasing the recursive compositional property that emerges at the semantic level.

Utility Function A function that scores candidate solutions, assessing their effectiveness in solving a task for a given input query. The goal of an AI system is to produce an output with high utility.

Value Model A model that estimates the expected utility of partial trajectories during a decoding process, guiding decoding strategies towards regions of expected high utility.

Decoding The process of extracting high-utility sequences or trajectories, that can be performed at both the syntactic and semantic levels.

Syntactic Decoding The process of extracting high-utility sequences of syntactic tokens, often relying on sampling tokens guided by value models or heuristics. Examples include top-p, value-guided beam search, and Monte Carlo tree search decoding.

Semantic Decoding The process of extracting high-utility trajectories in semantic space, leveraging sampling, semantic-level value models, or heuristics to optimize the output.

Examples include methods that orchestrate interactions between semantic processors, such as chain-of-thought, tree-of-thoughts, ReAct, and FunSearch. When abstracting the computational details, a semantic decoding algorithm becomes a semantic processor, showcasing the recursive compositional property that emerges at the semantic level.

Flows An abstract model of communication between LLMs, humans, and tools that supports the implementation of modular and compositional semantic decoding algorithms. Flows *are* semantic decoding algorithms in that they specify a general framework to implement them. We use the term *semantic decoding algorithm* to emphasize the optimization aspect in semantic space, while we use the term *Flows* to focus on the engineering and implementation aspects.

Bibliography

[1] Yelaman Abdullin et al. *Synthetic Dialogue Dataset Generation using LLM Agents*. 2024. arXiv: 2401.17461 [cs.CL].

[2] Alan Akbik et al. "FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 54–59. DOI: 10.18653/v1/N19-4010. URL: https://aclanthology.org/N19-4010.

[3] Saleema Amershi et al. "Guidelines for Human-AI Interaction". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*. Ed. by Stephen A. Brewster et al. ACM, 2019, p. 3. DOI: 10.1145/3290605.3300233. URL: https://doi.org/10.1145/3290605.3300233.

[4] Ateret Anaby-Tavor et al. "Do not have enough data? Deep learning to the rescue!" In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 7383–7390.

[5] John R. Anderson. *Cognitive Psychology and Its Implications*. 6th ed. New York: Worth Publishers, 2005, p. 519.

[6] Peter Anderson et al. "Guided Open Vocabulary Image Captioning with Constrained Beam Search". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 936–945. DOI: 10.18653/v1/D17-1098. URL: https://aclanthology.org/D17-1098.

[7] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. "Leveraging Linguistic Structure For Open Domain Information Extraction". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 344–354. DOI: 10.3115/v1/P15-1034. URL: https://aclanthology.org/P15-1034.

[8] Gabor Angeli et al. "Bootstrapped Self Training for Knowledge Base Population". In: *Proceedings of the 2015 Text Analysis Conference*. 2015.

[9] Joe Armstrong. “Making reliable distributed systems in the presence of software errors”. PhD thesis. Royal Institute of Technology, Stockholm, Sweden, 2003. URL: https://nbn-resolving.org/urn:nbn:se:kth:diva-3658.

[10] Akhil Arora, Alberto Garcia-Duran, and Robert West. “Low-Rank Subspaces for Unsupervised Entity Linking”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 8037–8054. DOI: 10.18653/v1/2021.emnlp-main.634. URL: https://aclanthology.org/2021.emnlp-main.634.

[11] Saleh Ashkboos et al. *QUIK: Towards End-to-End 4-Bit Inference on Generative Large Language Models*. 2023. arXiv: 2310.09259 [cs.LG].

[12] Dzmitry Bahdanau et al. “An Actor-Critic Algorithm for Sequence Prediction”. In: *CoRR* abs/1607.07086 (2016). arXiv: 1607.07086. URL: http://arxiv.org/abs/1607.07086.

[13] Amos Bairoch and Rolf Apweiler. “The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000”. In: *Nucleic Acids Research* 28.1 (Jan. 2000), pp. 45–48. ISSN: 0305-1048. DOI: 10.1093/nar/28.1.45. eprint: https://academic.oup.com/nar/article-pdf/28/1/45/9895197/280045.pdf. URL: https://doi.org/10.1093/nar/28.1.45.

[14] Laura Banarescu et al. “Abstract meaning representation for sembanking”. In: *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*. 2013, pp. 178–186.

[15] Michele Banko et al. “Open Information Extraction from the Web”. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence*. IJCAI’07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 2670–2676.

[16] Jie Bao et al. “Towards a theory of semantic communication”. In: *2011 IEEE Network Science Workshop*. 2011, pp. 110–117. DOI: 10.1109/NSW.2011.6004632.

[17] Maciej Besta et al. *Graph of Thoughts: Solving Elaborate Problems with Large Language Models*. 2023. arXiv: 2308.09687 [cs.CL].

[18] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. “Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation”. In: *arXiv preprint arXiv:2403.06988* (2024).

[19] Luca Beurer-Kellner, Marc Fischer, and Martin T. Vechev. “Prompting Is Programming: A Query Language for Large Language Models”. In: *Proc. ACM Program. Lang.* 7.PLDI (2023), pp. 1946–1969. DOI: 10.1145/3591300. URL: https://doi.org/10.1145/3591300.

[20] BIG-bench collaboration. “Beyond the Imitation Game: Measuring and extrapolating the capabilities of language models”. In: *In preparation* (2021). URL: https://github.com/google/BIG-bench/.

[21] Betty J Birner. *Introduction to pragmatics*. John Wiley & Sons, 2012.

[22] Ondrej Bojar et al. “Findings of the 2014 Workshop on Statistical Machine Translation”. In: *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Baltimore, Maryland, USA: Association for Computational Linguistics, June 2014, pp. 12–58. URL: http://www.aclweb.org/anthology/W/W14/W14-3302.

[23] Samuel Broscheit, Kiril Gashteovski, and Martin Achenbach. “OpenIE for Slot Filling at TAC KBP 2017 - System Description”. In: *proceedings of the Text Analysis Conference*. NIST, 2017.

[24] Tom Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[25] Collin Burns et al. *Weak-to-Strong Generalization: Eliciting Strong Capabilities With Weak Supervision*. 2023. arXiv: 2312.09390 [cs.CL].

[26] Zefan Cai, Baobao Chang, and Wenjuan Han. *Human-in-the-Loop through Chain-of-Thought*. 2023. arXiv: 2306.07932 [cs.CL].

[27] Rudolf Carnap and Yehoshua Bar-Hillel. “An Outline of a Theory of Semantic Information”. In: *Journal of Symbolic Logic* 19.3 (1954), pp. 230–232. DOI: 10.2307/2268645.

[28] Thiago Castro Ferreira et al. “The 2020 Bilingual, Bi-Directional WebNLG+ Shared Task: Overview and Evaluation Results (WebNLG+ 2020)”. In: *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*. Dublin, Ireland (Virtual): Association for Computational Linguistics, Dec. 2020, pp. 55–76. URL: https://aclanthology.org/2020.webnlg-1.7.

[29] Antoine Chaffin, Vincent Claveau, and Ewa Kijak. “PPL-MCTS: Constrained Textual Generation Through Discriminator-Guided MCTS Decoding”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 2953–2967. DOI: 10.18653/v1/2022.naacl-main.215. URL: https://aclanthology.org/2022.naacl-main.215.

[30] Arun Chaganty et al. “Importance sampling for unbiased on-demand evaluation of knowledge base population”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 1038–1048. DOI: 10.18653/v1/D17-1109. URL: https://aclanthology.org/D17-1109.

[31] Jun Shern Chan et al. “MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering”. In: *arXiv preprint arXiv:2410.07095* (2024).

[32] Daniel Chandler. *Semiotics: the basics*. Routledge, 2022.

[33] Harrison Chase. *LangChain*. https://github.com/hwchase17/langchain. 2022.

[34] Baian Chen et al. *FireAct: Toward Language Agent Fine-tuning*. 2023. arXiv: 2310.05915 [cs.CL].

[35] Charlie Chen et al. *Accelerating Large Language Model Decoding with Speculative Sampling*. 2023. arXiv: 2302.01318 [cs.CL].

[36] Justin Chih-Yao Chen et al. *MAGDi: Structured Distillation of Multi-Agent Interaction Graphs Improves Reasoning in Smaller Language Models*. 2024. arXiv: 2402.01620 [cs.CL].

[37] Mark Chen et al. "Evaluating Large Language Models Trained on Code". In: *ArXiv* abs/2107.03374 (2021).

[38] Wenhu Chen et al. "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks". In: *ArXiv* abs/2211.12588 (2022).

[39] Wenhu Chen et al. *Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks*. 2023. arXiv: 2211.12588 [cs.CL].

[40] Xinyun Chen et al. "Teaching Large Language Models to Self-Debug". In: *ArXiv* abs/2304.05128 (2023).

[41] Zixiang Chen et al. *Self-Play Fine-Tuning Converts Weak Language Models to Strong Language Models*. 2024. arXiv: 2401.01335 [cs.LG].

[42] Aakanksha Chowdhery et al. "PaLM: Scaling Language Modeling with Pathways". In: *CoRR* abs/2204.02311 (2022). DOI: 10.48550/arXiv.2204.02311. arXiv: 2204.02311. URL: https://doi.org/10.48550/arXiv.2204.02311.

[43] Hyung Won Chung et al. "Scaling Instruction-Finetuned Language Models". In: *CoRR* abs/2210.11416 (2022). DOI: 10.48550/arXiv.2210.11416. arXiv: 2210.11416. URL: https://doi.org/10.48550/arXiv.2210.11416.

[44] Kevin Clark and Christopher D. Manning. "Deep Reinforcement Learning for Mention-Ranking Coreference Models". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2256–2262. DOI: 10.18653/v1/D16-1245. URL: https://aclanthology.org/D16-1245.

[45] Ronan Collobert, Awni Hannun, and Gabriel Synnaeve. "A fully differentiable beam search decoder". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 2019, pp. 1341–1350. URL: https://proceedings.mlr.press/v97/collobert19a.html.

[46] SGL Project Contributors. *SGLang*. https://github.com/sgl-project/sglang. 2023.

[47] Luciano Del Corro and Rainer Gemulla. "ClausIE: clause-based open information extraction". In: *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*. Ed. by Daniel Schwabe et al. International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 355–366. DOI: 10.1145/2488388.2488420. URL: https://doi.org/10.1145/2488388.2488420.

[48] Luciano Del Corro et al. *SkipDecode: Autoregressive Skip Decoding with Batching and Caching for Efficient LLM Inference*. 2023. arXiv: 2307.02628 [cs.CL].

[49] Tri Dao et al. “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by Sanmi Koyejo et al. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

[50] Ishita Dasgupta et al. *Collaborating with language models for embodied reasoning*. 2023. arXiv: 2302.00763 [cs.LG].

[51] Nicola De Cao, Wilker Aziz, and Ivan Titov. “Highly Parallel Autoregressive Entity Linking with Discriminative Correction”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 7662–7669. DOI: 10.18653/v1/2021.emnlp-main.604. URL: https://aclanthology.org/2021.emnlp-main.604.

[52] Nicola De Cao et al. “Autoregressive Entity Retrieval”. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=5k8F6UU39V.

[53] Nicola De Cao et al. “Multilingual Autoregressive Entity Linking”. In: *Transactions of the Association for Computational Linguistics* 10 (Mar. 2022), pp. 274–290. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00460. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00460/2004070/tacl_a_00460.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00460.

[54] Nicola De Cao et al. “Multilingual Autoregressive Entity Linking”. In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 274–290. DOI: 10.1162/tacl_a_00460. URL: https://aclanthology.org/2022.tacl-1.16.

[55] Tim Dettmers et al. *LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale*. 2022. arXiv: 2208.07339 [cs.LG].

[56] Tim Dettmers et al. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023. arXiv: 2305.14314 [cs.LG].

[57] Daniel Deutsch, Shyam Upadhyay, and Dan Roth. “A General-Purpose Algorithm for Constrained Sequential Inference”. In: *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 482–492. DOI: 10.18653/v1/K19-1045. URL: https://aclanthology.org/K19-1045.

[58] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

[59] Ruomeng Ding et al. *Everything of Thoughts: Defying the Law of Penrose Triangle for Thought Generation*. 2023. arXiv: 2311.04254 [cs.AI].

[60] Pierre Dognin et al. "ReGen: Reinforcement Learning for Text and Knowledge Base Generation using Pretrained Language Models". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 1084–1099. DOI: 10.18653/v1/2021.emnlp-main.83. URL: https://aclanthology.org/2021.emnlp-main.83.

[61] Yu Du, Fangyun Wei, and Hongyang Zhang. *AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls*. 2024. arXiv: 2402.04253 [cs.CL].

[62] Ahmed Elnaggar et al. "ProtTrans: Towards Cracking the Language of Lifes Code Through Self-Supervised Deep Learning and High Performance Computing". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3095381.

[63] Hady Elsahar et al. "T-REx: A Large Scale Alignment of Natural Language with Knowledge Base Triples". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: https://aclanthology.org/L18-1544.

[64] Reliable ETH Secure and Intelligent Systems. *LMQL*. https://github.com/eth-sri/lmql. 2023.

[65] Angela Fan, Mike Lewis, and Yann Dauphin. "Hierarchical Neural Story Generation". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 889–898. DOI: 10.18653/v1/P18-1082. URL: https://aclanthology.org/P18-1082.

[66] Chrisantha Fernando et al. *Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution*. 2023. arXiv: 2309.16797 [cs.CL].

[67] Chrisantha Fernando et al. "Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution". In: *CoRR* abs/2309.16797 (2023). DOI: 10.48550/ARXIV.2309.16797. arXiv: 2309.16797. URL: https://doi.org/10.48550/arXiv.2309.16797.

[68] Noelia Ferruz, Steffen Schmidt, and Birte Höcker. "ProtGPT2 is a deep unsupervised language model for protein design". In: *Nature Communications* 13 (2022).

[69] Luciano Floridi. "Philosophical Conceptions of Information". In: *Lecture Notes in Computer Science - LNCS* 5363 (Jan. 2009), pp. 13–53. DOI: 10.1007/978-3-642-00659-3_2.

[70] Jerry A. Fodor. *The Language of Thought*. Harvard University Press, 1975.

[71] Elias Frantar et al. *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*. 2023. arXiv: 2210.17323 [cs.LG].

[72] Philip Gage. "A new algorithm for data compression". In: *C Users J.* 12.2 (Feb. 1994), pp. 23–38. ISSN: 0898-9788.

[73] Luis Galárraga et al. "Canonicalizing Open Knowledge Bases". In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*. Ed. by Jianzhong Li et al. ACM, 2014, pp. 1679–1688. DOI: 10.1145/2661829.2662073. URL: https://doi.org/10.1145/2661829.2662073.

[74] Isabel O. Gallegos et al. *Self-Debiasing Large Language Models: Zero-Shot Recognition and Reduction of Stereotypes*. 2024. arXiv: 2402.01981 [cs.CL].

[75] Octavian-Eugen Ganea and Thomas Hofmann. "Deep Joint Entity Disambiguation with Local Neural Attention". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 2619–2629. DOI: 10.18653/v1/D17-1277. URL: https://aclanthology.org/D17-1277.

[76] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. "PPDB: The Paraphrase Database". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 758–764. URL: https://aclanthology.org/N13-1092.

[77] Jiahui Gao et al. *ZeroGen$^+$: Self-Guided High-Quality Data Generation in Efficient Zero-Shot Learning*. 2022. DOI: 10.48550/ARXIV.2205.12679. URL: https://arxiv.org/abs/2205.12679.

[78] Luyu Gao et al. *PAL: Program-aided Language Models*. 2023. arXiv: 2211.10435 [cs.CL].

[79] Tianyu Gao, Adam Fisch, and Danqi Chen. "Making Pre-trained Language Models Better Few-shot Learners". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 3816–3830. DOI: 10.18653/v1/2021.acl-long.295. URL: https://aclanthology.org/2021.acl-long.295.

[80] David Garlan and Mary Shaw. "An Introduction to Software Architecture". In: *Advances in Software Engineering and Knowledge Engineering*. Ed. by Vincenzo Ambriola and Genoveffa Tortora. Vol. 2. Series on Software Engineering and Knowledge Engineering. World Scientific, 1993, pp. 1–39. DOI: 10.1142/9789812798039_0001. URL: https://doi.org/10.1142/9789812798039%5C_0001.

[81] Kiril Gashteovski, Rainer Gemulla, and Luciano del Corro. "MinIE: Minimizing Facts in Open Information Extraction". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 2630–2640. DOI: 10.18653/v1/D17-1278. URL: https://aclanthology.org/D17-1278.

[82] Kiril Gashteovski et al. “On Aligning OpenIE Extractions with Knowledge Bases: A Case Study”. In: *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*. Online: Association for Computational Linguistics, Nov. 2020, pp. 143–154. DOI: 10.18653/v1/2020.eval4nlp-1.14. URL: https://aclanthology.org/2020.eval4nlp-1.14.

[83] Samuel Gehman et al. “RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 3356–3369. DOI: 10.18653/v1/2020.findings-emnlp.301. URL: https://aclanthology.org/2020.findings-emnlp.301.

[84] Atticus Geiger et al. “Faithful, Interpretable Model Explanations via Causal Abstraction”. Stanford AI Lab Blog. 2022. URL: https://ai.stanford.edu/blog/causal-abstraction/.

[85] Saibo Geng et al. “Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 10932–10952. DOI: 10.18653/v1/2023.emnlp-main.674. URL: https://aclanthology.org/2023.emnlp-main.674.

[86] Saibo Geng et al. “Sketch-Guided Constrained Decoding for Boosting Blackbox Large Language Models without Logit Access”. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 234–245. DOI: 10.18653/v1/2024.acl-short.23. URL: https://aclanthology.org/2024.acl-short.23.

[87] Shahriar Golchin and Mihai Surdeanu. *Time Travel in LLMs: Tracing Data Contamination in Large Language Models*. 2024. arXiv: 2308.08493 [cs.CL].

[88] Google. *Gemini API reference*. https://ai.google.dev/api?lang=python. Accessed: 28 October 2024. 2023.

[89] Adam Grycner and Gerhard Weikum. “POLY: Mining Relational Paraphrases from Multilingual Sentences”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2183–2192. DOI: 10.18653/v1/D16-1236. URL: https://aclanthology.org/D16-1236.

[90] Jiatao Gu et al. “Non-Autoregressive Neural Machine Translation”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=B1l8BtlCb.

[91] Çaglar Gülçehre et al. “Reinforced Self-Training (ReST) for Language Modeling”. In: *CoRR* abs/2308.08998 (2023). DOI: 10.48550/ARXIV.2308.08998. arXiv: 2308.08998. URL: https://doi.org/10.48550/arXiv.2308.08998.

[92] Qipeng Guo et al. “CycleGT: Unsupervised Graph-to-Text and Text-to-Graph Generation via Cycle Training”. In: *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*. Dublin, Ireland (Virtual): Association for Computational Linguistics, Dec. 2020, pp. 77–88. URL: https://aclanthology.org/2020.webnlg-1.8.

[93] Xu Han et al. “FewRel: A Large-Scale Supervised Few-Shot Relation Classification Dataset with State-of-the-Art Evaluation”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 4803–4809. DOI: 10.18653/v1/D18-1514. URL: https://aclanthology.org/D18-1514.

[94] Laura Hanu and Unitary team. *Detoxify*. Github. https://github.com/unitaryai/detoxify. 2020.

[95] Di He et al. “Decoding with Value Networks for Neural Machine Translation”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/2b24d495052a8ce66358eb576b8912c8-Paper.pdf.

[96] Junxian He et al. “Towards a Unified View of Parameter-Efficient Transfer Learning”. In: *CoRR* abs/2110.04366 (2021). URL: https://arxiv.org/abs/2110.04366.

[97] Dan Hendrycks et al. “Measuring Coding Challenge Competence With APPS”. In: *NeurIPS* (2021).

[98] Carl E. Hewitt. “Actor Model of Computation: Scalable Robust Information Systems”. In: *arXiv: Programming Languages* (2010).

[99] Carl E. Hewitt, Peter Boehler Bishop, and Richard Steiger. “A Universal Modular ACTOR Formalism for Artificial Intelligence”. In: *International Joint Conference on Artificial Intelligence*. 1973.

[100] Johannes Hoffart et al. “Robust Disambiguation of Named Entities in Text”. In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, July 2011, pp. 782–792. URL: https://aclanthology.org/D11-1072.

[101] Chris Hokamp and Qun Liu. “Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 1535–1546. DOI: 10.18653/v1/P17-1141. URL: https://aclanthology.org/P17-1141.

[102] Ari Holtzman et al. “The Curious Case of Neural Text Degeneration”. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=rygGQyrFvH.

[103] Sirui Hong et al. “MetaGPT: Meta Programming for Multi-Agent Collaborative Framework”. In: *CoRR* abs/2308.00352 (2023). DOI: 10.48550/ARXIV.2308.00352. arXiv: 2308.00352. URL: https://doi.org/10.48550/arXiv.2308.00352.

[104] Or Honovich et al. “Unnatural Instructions: Tuning Language Models with (Almost) No Human Labor”. In: *arXiv preprint arXiv:2212.09689* (2022).

[105] Eric Horvitz. “Principles of Mixed-Initiative User Interfaces”. In: *Proceeding of the CHI '99 Conference on Human Factors in Computing Systems: The CHI is the Limit, Pittsburgh, PA, USA, May 15-20, 1999*. Ed. by Marian G. Williams and Mark W. Altom. ACM, 1999, pp. 159–166. DOI: 10.1145/302979.303030. URL: https://doi.org/10.1145/302979.303030.

[106] Eric Horvitz. “Principles of mixed-initiative user interfaces”. In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 1999, pp. 159–166.

[107] Bin Hu et al. *Enabling Intelligent Interactions between an Agent and an LLM: A Reinforcement Learning Approach*. 2023. arXiv: 2306.03604 [cs.AI].

[108] J. Edward Hu et al. “Improved Lexically Constrained Decoding for Translation and Monolingual Rewriting”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 839–850. DOI: 10.18653/v1/N19-1090. URL: https://aclanthology.org/N19-1090.

[109] Hongzhao Huang, Larry Heck, and Heng Ji. “Leveraging deep neural networks and knowledge graphs for entity disambiguation”. In: *ArXiv preprint* abs/1504.07678 (2015). URL: https://arxiv.org/abs/1504.07678.

[110] Jie Huang et al. “Large Language Models Cannot Self-Correct Reasoning Yet”. In: *CoRR* abs/2310.01798 (2023). DOI: 10.48550/ARXIV.2310.01798. arXiv: 2310.01798. URL: https://doi.org/10.48550/arXiv.2310.01798.

[111] Pere-Lluís Huguet Cabot and Roberto Navigli. “REBEL: Relation Extraction By End-to-end Language generation”. In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2370–2381. DOI: 10.18653/v1/2021.findings-emnlp.204. URL: https://aclanthology.org/2021.findings-emnlp.204.

[112] Julian Ibarz et al. “How to train your robot with deep reinforcement learning: lessons we have learned”. In: *The International Journal of Robotics Research* 40.4-5 (2021), pp. 698–721.

[113] Heng Ji and Ralph Grishman. “Knowledge Base Population: Successful Approaches and Challenges”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 1148–1158. URL: https://aclanthology.org/P11-1115.

[114] Shaoxiong Ji et al. “A Survey on Knowledge Graphs: Representation, Acquisition and Applications”. In: *IEEE transactions on neural networks and learning systems* (2021). DOI: 10.1109/TNNLS.2021.3070843. URL: https://ieeexplore.ieee.org/abstract/document/9416312?casa_token=0i6WvLC-GDAAAAAA:IrR5LNlVlhYIchmQc_tEyqsvgiQIBBHjD9rvLwX8SdqsmGo1

[115] Aoran Jiao et al. *Swarm-GPT: Combining Large Language Models with Safe Motion Planning for Robot Choreography Design*. 2023. arXiv: 2312.01059 [cs.RO].

[116] Carlos E Jimenez et al. “SWE-bench: Can Language Models Resolve Real-world Github Issues?” In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=VTF8yNQM66.

[117] Martin Josifoski et al. “Exploiting Asymmetry for Synthetic Training Data Generation: SynthIE and the Case of Information Extraction”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1555–1574. DOI: 10.18653/v1/2023.emnlp-main.96. URL: https://aclanthology.org/2023.emnlp-main.96.

[118] Martin Josifoski et al. *Flows: Building Blocks of Reasoning and Collaborating AI*. 2023. arXiv: 2308.01285 [cs.AI].

[119] Martin Josifoski et al. “GenIE: Generative Information Extraction”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 4626–4643. DOI: 10.18653/v1/2022.naacl-main.342. URL: https://aclanthology.org/2022.naacl-main.342.

[120] Martin Josifoski et al. “Language Model Decoding as Likelihood–Utility Alignment”. In: *Findings of the Association for Computational Linguistics: EACL 2023*. Ed. by Andreas Vlachos and Isabelle Augenstein. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 1455–1470. DOI: 10.18653/v1/2023.findings-eacl.107. URL: https://aclanthology.org/2023.findings-eacl.107.

[121] John Jumper et al. “Highly accurate protein structure prediction with AlphaFold”. In: *Nature* 596.7873 (2021), pp. 583–589. DOI: 10.1038/s41586-021-03819-2.

[122] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. “Language Models can Solve Computer Tasks”. In: *ArXiv* abs/2303.17491 (2023).

[123] Hannah Kim et al. *MEGAnno+: A Human-LLM Collaborative Annotation System*. 2024. arXiv: 2402.18050 [cs.CL].

[124] Sehoon Kim et al. *Speculative Decoding with Big Little Decoder*. 2023. arXiv: 2302.07863 [cs.CL].

[125] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *Proceedings of the 3rd International Conference on Learning Representations (ICLR)* (2015). URL: https://arxiv.org/abs/1412.6980.

[126] Guillaume Klein et al. "OpenNMT: Open-Source Toolkit for Neural Machine Translation". In: *Proceedings of ACL 2017, System Demonstrations*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 67–72. URL: https://aclanthology.org/P17-4012.

[127] Takeshi Kojima et al. "Large Language Models are Zero-Shot Reasoners". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 22199–22213. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/8bb0d291acd4acf06ef112099c16f326-Paper-Conference.pdf.

[128] Nikolaos Kolitsas, Octavian-Eugen Ganea, and Thomas Hofmann. "End-to-End Neural Entity Linking". In: *Proceedings of the 22nd Conference on Computational Natural Language Learning*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 519–529. DOI: 10.18653/v1/K18-1050. URL: https://aclanthology.org/K18-1050.

[129] Wouter Kool, Herke Van Hoof, and Max Welling. "Stochastic Beams and Where To Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, July 2019, pp. 3499–3508. URL: https://proceedings.mlr.press/v97/kool19a.html.

[130] Kalpesh Krishna et al. "RankGen: Improving Text Generation with Large Ranking Models". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 199–232. URL: https://aclanthology.org/2022.emnlp-main.15.

[131] Varun Kumar, Ashutosh Choudhary, and Eunah Cho. "Data augmentation using pre-trained transformer models". In: *arXiv preprint arXiv:2003.02245* (2020).

[132] EPFL Data Science Lab. *transformers-CFG*. https://github.com/epfl-dlab/transformers-CFG. 2023.

[133] John E. Laird. *The Soar Cognitive Architecture*. The MIT Press, 2012. ISBN: 0262122960.

[134] Brenden M. Lake et al. "Building Machines That Learn and Think Like People". In: *CoRR* abs/1604.00289 (2016). arXiv: 1604.00289. URL: http://arxiv.org/abs/1604.00289.

[135] Zhenzhong Lan et al. "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=H1eA7AEtvS.

[136] Phong Le and Ivan Titov. "Improving Entity Linking by Modeling Latent Relations between Mentions". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 1595–1604. DOI: 10.18653/v1/P18-1148. URL: https://aclanthology.org/P18-1148.

[137] Rémi Leblond et al. “Machine Translation Decoding beyond Beam Search”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 8410–8434. DOI: 10.18653/v1/2021.emnlp-main.662. URL: https://aclanthology.org/2021.emnlp-main.662.

[138] Chia-Hsuan Lee, Hao Cheng, and Mari Ostendorf. *OrchestraLLM: Efficient Orchestration of Language Models for Dialogue State Tracking*. 2023. arXiv: 2311.09758 [cs.CL].

[139] LeetCode. *Leetcode.com*. 2023. URL: https://leetcode.com (visited on 05/15/2023).

[140] Mike Lewis et al. “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703. URL: https://aclanthology.org/2020.acl-main.703.

[141] Chengshu Li et al. *Chain of Code: Reasoning with a Language Model-Augmented Code Emulator*. 2023. arXiv: 2312.04474 [cs.CL].

[142] Guohao Li et al. “Camel: Communicative agents for" mind" exploration of large scale language model society”. In: *arXiv preprint arXiv:2303.17760* (2023).

[143] Junlong Li, Zhuosheng Zhang, and Hai Zhao. *Self-Prompting Large Language Models for Open-Domain QA*. 2022. DOI: 10.48550/ARXIV.2212.08635. URL: https://arxiv.org/abs/2212.08635.

[144] Liunian Harold Li et al. “Symbolic Chain-of-Thought Distillation: Small Models Can Also “Think” Step-by-Step”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 2665–2679. DOI: 10.18653/v1/2023.acl-long.150. URL: https://aclanthology.org/2023.acl-long.150.

[145] Qintong Li et al. *Collaborative Evaluation: Exploring the Synergy of Large Language Models and Humans for Open-ended Generation Evaluation*. 2023. arXiv: 2310.19740 [cs.CL].

[146] Yucheng Li, Frank Guerin, and Chenghua Lin. *LatestEval: Addressing Data Contamination in Language Model Evaluation through Dynamic and Time-Sensitive Test Construction*. 2023. arXiv: 2312.12343 [cs.CL].

[147] Yujia Li et al. “Competition-level code generation with alphacode”. In: *Science* 378.6624 (2022), pp. 1092–1097.

[148] Yankai Lin et al. “Neural Relation Extraction with Selective Attention over Instances”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 2124–2133. DOI: 10.18653/v1/P16-1200. URL: https://aclanthology.org/P16-1200.

[149] Zachary C. Lipton. "The Mythos of Model Interpretability". In: *Commun. ACM* 61.10 (Sept. 2018), pp. 36–43. ISSN: 0001-0782. DOI: 10.1145/3233231. URL: https://doi.org/10.1145/3233231.

[150] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *ArXiv preprint* abs/1907.11692 (2019). URL: https://arxiv.org/abs/1907.11692.

[151] Yue Liu et al. "Seq2RDF: An End-to-end Application for Deriving Triples from Natural Language Text". In: *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018*. 2018.

[152] Zijun Liu et al. *Dynamic LLM-Agent Network: An LLM-agent Collaboration Framework with Agent Team Optimization*. 2023. arXiv: 2310.02170 [cs.CL].

[153] Jieyi Long. *Large Language Model Guided Tree-of-Thought*. 2023. arXiv: 2305.08291 [cs.AI].

[154] Pan Lu et al. "Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models". In: *ArXiv* abs/2304.09842 (2023).

[155] Ximing Lu et al. "NeuroLogic A*esque Decoding: Constrained Text Generation with Lookahead Heuristics". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 780–799. DOI: 10.18653/v1/2022.naacl-main.57. URL: https://aclanthology.org/2022.naacl-main.57.

[156] Zimu Lu et al. *MathGenie: Generating Synthetic Data with Question Back-translation for Enhancing Mathematical Reasoning of LLMs*. 2024. arXiv: 2402.16352 [cs.CL].

[157] Aman Madaan et al. "Self-refine: Iterative refinement with self-feedback". In: *arXiv preprint arXiv:2303.17651* (2023).

[158] Inbal Magar and Roy Schwartz. "Data Contamination: From Memorization to Exploitation". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Association for Computational Linguistics, 2022, pp. 157–165. DOI: 10.18653/v1/2022.acl-short.18. URL: https://doi.org/10.18653/v1/2022.acl-short.18.

[159] Lucie Charlotte Magister et al. "Teaching Small Language Models to Reason". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 1773–1781. DOI: 10.18653/v1/2023.acl-short.151. URL: https://aclanthology.org/2023.acl-short.151.

[160] Clara Meister, Ryan Cotterell, and Tim Vieira. "If beam search is the answer, what was the question?" In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 2173–2185. DOI: 10.18653/v1/2020.emnlp-main.170. URL: https://aclanthology.org/2020.emnlp-main.170.

[161] Clara Meister et al. "Conditional Poisson Stochastic Beams". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 664–681. DOI: 10.18653/v1/2021.emnlp-main.52. URL: https://aclanthology.org/2021.emnlp-main.52.

[162] Clara Meister et al. "Locally Typical Sampling". In: *Transactions of the Association for Computational Linguistics* 11 (Jan. 2023), pp. 102–121. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00536. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00536/2067865/tacl_a_00536.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00536.

[163] Clara Meister et al. "On the Efficacy of Sampling Adapters". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 1437–1455. DOI: 10.18653/v1/2023.acl-long.80. URL: https://aclanthology.org/2023.acl-long.80.

[164] Clara Meister et al. "On the probability-quality paradox in language generation". In: *CoRR* abs/2203.17217 (2022). DOI: 10.48550/arXiv.2203.17217. arXiv: 2203.17217. URL: https://doi.org/10.48550/arXiv.2203.17217.

[165] Yu Meng et al. *Generating Training Data with Language Models: Towards Zero-Shot Language Understanding*. 2022. DOI: 10.48550/ARXIV.2202.04538. URL: https://arxiv.org/abs/2202.04538.

[166] Yu Meng et al. "Generating training data with language models: Towards zero-shot language understanding". In: *arXiv preprint arXiv:2202.04538* (2022).

[167] Yu Meng et al. *Tuning Language Models as Training Data Generators for Augmentation-Enhanced Few-Shot Learning*. 2022. DOI: 10.48550/ARXIV.2211.03044. URL: https://arxiv.org/abs/2211.03044.

[168] Filipe Mesquita et al. "KnowledgeNet: A Benchmark Dataset for Knowledge Base Population". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 749–758. DOI: 10.18653/v1/D19-1069. URL: https://aclanthology.org/D19-1069.

[169] Grégoire Mialon et al. "Augmented Language Models: a Survey". In: *CoRR* abs/2302.07842 (2023). DOI: 10.48550/ARXIV.2302.07842. arXiv: 2302.07842. URL: https://doi.org/10.48550/arXiv.2302.07842.

[170] Grégoire Mialon et al. *GAIA: a benchmark for General AI Assistants*. 2023. arXiv: 2311.12983 [cs.CL].

[171] Microsoft. *Guidance*. https://github.com/guidance-ai/guidance. 2023.

[172] Scott Miller et al. "Algorithms That Learn to Extract Information BBN: TIPSTER Phase III". In: *TIPSTER TEXT PROGRAM PHASE III: Proceedings of a Workshop held at Baltimore, Maryland, October 13-15, 1998*. Baltimore, Maryland, USA: Association for Computational Linguistics, 1998, pp. 75–89. DOI: 10.3115/1119089.1119107. URL: https://aclanthology.org/X98-1014.

[173] David Milne and Ian H. Witten. "Learning to Link with Wikipedia". In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. CIKM '08. Napa Valley, California, USA: Association for Computing Machinery, 2008, pp. 509–518. ISBN: 9781595939913. DOI: 10.1145/1458082.1458150. URL: https://doi.org/10.1145/1458082.1458150.

[174] Mike Mirzayanov. *Codeforces.com*. 2023. URL: https://codeforces.com (visited on 05/15/2023).

[175] Biswesh Mohapatra et al. "Simulated Chats for Building Dialog Systems: Learning to Generate Conversations from Instructions". In: *arXiv preprint arXiv:2010.10216* (2020).

[176] Hussein Mozannar et al. "When to Show a Suggestion? Integrating Human Feedback in AI-Assisted Programming". In: *CoRR* abs/2306.04930 (2023). DOI: 10.48550/arXiv.2306.04930. arXiv: 2306.04930. URL: https://doi.org/10.48550/arXiv.2306.04930.

[177] Yohei Nakajima. *BabyAGI*. https://github.com/yoheinakajima/babyagi. 2023.

[178] Reiichiro Nakano et al. "WebGPT: Browser-assisted question-answering with human feedback". In: *CoRR* abs/2112.09332 (2021). arXiv: 2112.09332. URL: https://arxiv.org/abs/2112.09332.

[179] Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. "PATTY: A Taxonomy of Relational Patterns with Semantic Types". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, July 2012, pp. 1135–1145. URL: https://aclanthology.org/D12-1104.

[180] Tapas Nayak and Hwee Tou Ng. "Effective Modeling of Encoder-Decoder Architecture for Joint Entity and Relation Extraction". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.05 (Apr. 2020), pp. 8528–8535. DOI: 10.1609/aaai.v34i05.6374. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6374.

[181] Nils J. Nilsson. *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge: Cambridge University Press, 2009. ISBN: 9780521122931.

[182] Xuefei Ning et al. *Skeleton-of-Thought: Large Language Models Can Do Parallel Decoding*. 2023. arXiv: 2307.15337 [cs.CL].

[183] Mohammad Norouzi et al. “Reward Augmented Maximum Likelihood for Neural Structured Prediction”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/2f885d0fbe2e131bfc9d98363e55d1d4-Paper.pdf.

[184] Maxwell I. Nye et al. “Show Your Work: Scratchpads for Intermediate Computation with Language Models”. In: *CoRR* abs/2112.00114 (2021). arXiv: 2112.00114. URL: https://arxiv.org/abs/2112.00114.

[185] OpenAI. *ChatGPT*. https://openai.com/chatgpt. Accessed: 2024-01-28. 2023.

[186] OpenAI. “GPT-4 Technical Report”. In: *ArXiv* abs/2303.08774 (2023).

[187] OpenAI. *OpenAI API Reference*. https://platform.openai.com/docs/api-reference/introduction. Accessed: 28 October 2024. 2023.

[188] Yannis Papanikolaou and Andrea Pierleoni. “Dare: Data augmented relation extraction with gpt-2”. In: *arXiv preprint arXiv:2004.13845* (2020).

[189] Kanghee Park et al. “Grammar-Aligned Decoding”. In: *arXiv preprint arXiv:2405.21047* (2024).

[190] Debjit Paul et al. “REFINER: Reasoning Feedback on Intermediate Representations”. In: *arXiv preprint arXiv:2304.01904* (2023). eprint: 2304.01904.

[191] Judea Pearl and Dana Mackenzie. *The Book of Why. The New Science of Cause and Effect*. New York: Basic Books, 2018. ISBN: 978-0-465-09760-9.

[192] Anastassis Perrakis, Richard J. Morris, and Victor S. Lamzin. “Automated protein model building combined with iterative structure refinement”. In: *Nature Structural Biology* 6 (1999), pp. 458–463. URL: https://api.semanticscholar.org/CorpusID:20292852.

[193] Fabio Petroni et al. “KILT: a Benchmark for Knowledge Intensive Language Tasks”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 2523–2544. DOI: 10.18653/v1/2021.naacl-main.200. URL: https://aclanthology.org/2021.naacl-main.200.

[194] Maxime Peyrard. “A Simple Theoretical Model of Importance for Summarization”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 1059–1073. DOI: 10.18653/v1/P19-1101. URL: https://aclanthology.org/P19-1101.

[195] Maxime Peyrard, Martin Josifoski, and Robert West. “The Era of Semantic Decoding”. In: *CoRR* abs/2403.14562 (2024). DOI: 10.48550/ARXIV.2403.14562. arXiv: 2403.14562. URL: https://doi.org/10.48550/arXiv.2403.14562.

[196] Gabriel Poesia et al. “Synchromesh: Reliable Code Generation from Pre-trained Language Models”. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL: https://openreview.net/forum?id=KmtVD97J43e.

[197] Stanislas Polu and Ilya Sutskever. “Generative Language Modeling for Automated Theorem Proving”. In: *CoRR* abs/2009.03393 (2020). arXiv: 2009.03393. URL: https://arxiv.org/abs/2009.03393.

[198] Reiner Pope et al. *Efficiently Scaling Transformer Inference*. 2022. arXiv: 2211.05102 [cs.LG].

[199] Matt Post and David Vilar. “Fast Lexically Constrained Decoding with Dynamic Beam Allocation for Neural Machine Translation”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 1314–1324. DOI: 10.18653/v1/N18-1119. URL: https://aclanthology.org/N18-1119.

[200] Angela Potochnik. *Idealization and the Aims of Science*. Chicago: University of Chicago Press, 2017.

[201] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: (2018). URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

[202] Rafael Rafailov et al. *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. 2023. arXiv: 2305.18290 [cs.LG].

[203] Guillem Ramírez et al. *Cache & Distil: Optimising API Calls to Large Language Models*. 2023. arXiv: 2310.13561 [cs.CL].

[204] Scott Reed et al. *A Generalist Agent*. 2022. arXiv: 2205.06175 [cs.AI].

[205] Machel Reid and Graham Neubig. “Learning to Model Editing Processes”. In: *Conference on Empirical Methods in Natural Language Processing*. 2022. URL: https://api.semanticscholar.org/CorpusID:249062636.

[206] Zhou Ren et al. “Deep Reinforcement Learning-Based Image Captioning with Embedding Reward”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 1151–1159. DOI: 10.1109/CVPR.2017.128. URL: https://doi.org/10.1109/CVPR.2017.128.

[207] Michael Rescorla. “The Language of Thought Hypothesis”. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta and Uri Nodelman. Winter 2023. Metaphysics Research Lab, Stanford University, 2023.

[208] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “"Why Should I Trust You?": Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778. URL: https://doi.org/10.1145/2939672.2939778.

[209] Toran Bruce Richards. *AutoGPT*. https://github.com/Significant-Gravitas/Auto-GPT. 2023.

[210] Robin Rombach et al. “High-Resolution Image Synthesis with Latent Diffusion Models”. In: *CoRR* abs/2112.10752 (2021). arXiv: 2112.10752. URL: https://arxiv.org/abs/2112.10752.

[211] Bernardino Romera-Paredes et al. “Mathematical discoveries from program search with large language models”. In: *Nature* (2023), pp. 1–3.

[212] Bernardino Romera-Paredes et al. “Mathematical discoveries from program search with large language models”. In: *Nature* (2023). DOI: 10.1038/s41586-023-06924-6.

[213] Subhro Roy et al. *BenchCLAMP: A Benchmark for Evaluating Language Models on Semantic Parsing*. 2022. arXiv: 2206.10668 [cs.CL].

[214] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN: 9780134610993. URL: http://aima.cs.berkeley.edu/.

[215] Chitwan Saharia et al. “Image Super-Resolution via Iterative Refinement”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45 (2021), pp. 4713–4726. URL: https://api.semanticscholar.org/CorpusID:233241040.

[216] Gaurav Sahu et al. *Data Augmentation for Intent Classification with Off-the-shelf Large Language Models*. 2022. DOI: 10.48550/ARXIV.2204.01959. URL: https://arxiv.org/abs/2204.01959.

[217] Ananya B. Sai, Akash Kumar Mohankumar, and Mitesh M. Khapra. “A Survey of Evaluation Metrics Used for NLG Systems”. In: 55.2 (2022). ISSN: 0360-0300. DOI: 10.1145/3485766. URL: https://doi.org/10.1145/3485766.

[218] Marija Šakota, Maxime Peyrard, and Robert West. “Fly-Swat or Cannon? Cost-Effective Language Model Choice via Meta-Modeling”. In: *Proceedings of The International ACM Conference on Web Search and Data Mining (WSDM)*. 2024.

[219] Timo Schick et al. “PEER: A Collaborative Language Model”. In: *ArXiv* abs/2208.11663 (2022). URL: https://api.semanticscholar.org/CorpusID:251765117.

[220] Timo Schick et al. “Toolformer: Language Models Can Teach Themselves to Use Tools”. In: *ArXiv* abs/2302.04761 (2023).

[221] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. “PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 9895–9901. DOI: 10.18653/v1/2021.emnlp-main.779. URL: https://aclanthology.org/2021.emnlp-main.779.

[222] Tal Schuster et al. “Confident Adaptive Language Modeling”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 17456–17472. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/6fac9e316a4ae75ea244ddcef1982c71-Paper-Conference.pdf.

[223] Bilgehan Sel et al. *Algorithm of Thoughts: Enhancing Exploration of Ideas in Large Language Models*. 2023. arXiv: 2308.10379 [cs.CL].

[224] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. URL: https://aclanthology.org/P16-1162.

[225] Claude Elwood Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27 (1948), pp. 379–423. URL: http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf (visited on 04/22/2003).

[226] Kun Shao et al. "A survey of deep reinforcement learning in video games". In: *arXiv preprint arXiv:1912.10944* (2019).

[227] Zhihong Shao et al. *Synthetic Prompting: Generating Chain-of-Thought Demonstrations for Large Language Models*. 2023. DOI: 10.48550/ARXIV.2302.00618. URL: https://arxiv.org/abs/2302.00618.

[228] Wei Shen, Jianyong Wang, and Jiawei Han. "Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions". In: *IEEE Transactions on Knowledge and Data Engineering* (2015).

[229] Yongliang Shen et al. "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in HuggingFace". In: *ArXiv* abs/2303.17580 (2023).

[230] Weijia Shi et al. *Detecting Pretraining Data from Large Language Models*. 2023. arXiv: 2310.16789 [cs.CL].

[231] Richard Shin et al. "Constrained Language Models Yield Few-Shot Semantic Parsers". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 7699–7715. DOI: 10.18653/v1/2021.emnlp-main.608. URL: https://aclanthology.org/2021.emnlp-main.608.

[232] Noah Shinn et al. "Reflexion: Language Agents with Verbal Reinforcement Learning". In: *arXiv preprint arXiv:2303.11366* (2023).

[233] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI].

[234] Avi Singh et al. *Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models*. 2023. arXiv: 2312.06585 [cs.LG].

[235] Ryan Smith et al. *Language Models in the Loop: Incorporating Prompting into Weak Supervision*. 2022. DOI: 10.48550/ARXIV.2205.02318. URL: https://arxiv.org/abs/2205.02318.

[236] Shaden Smith et al. "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model". In: *arXiv preprint arXiv:2201.11990* (2022).

[237] Yixin Song et al. *PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU*. 2023. arXiv: 2312.12456 [cs.LG].

[238] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[239] Felix Stahlberg and Bill Byrne. “On NMT Search Errors and Model Errors: Cat Got Your Tongue?” In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3356–3362. DOI: 10.18653/v1/D19-1331. URL: https://aclanthology.org/D19-1331.

[240] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. “Blockwise Parallel Decoding for Deep Autoregressive Models”. In: *CoRR* abs/1811.03115 (2018). arXiv: 1811.03115. URL: http://arxiv.org/abs/1811.03115.

[241] Yixuan Su et al. “A Contrastive Framework for Neural Text Generation”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=V88BafmH9Pj.

[242] Dianbo Sui et al. “Set Generation Networks for End-to-End Knowledge Base Population”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 9650–9660. DOI: 10.18653/v1/2021.emnlp-main.760. URL: https://aclanthology.org/2021.emnlp-main.760.

[243] Mihai Surdeanu. “Overview of the TAC2013 Knowledge Base Population Evaluation: English Slot Filling and Temporal Slot Filling”. In: *Theory and Applications of Categories* (2013).

[244] Ilya Sutskever, James Martens, and Geoffrey E. Hinton. “Generating Text with Recurrent Neural Networks”. In: *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*. Ed. by Lise Getoor and Tobias Scheffer. Omnipress, 2011, pp. 1017–1024. URL: https://icml.cc/2011/papers/524%5C_icmlpaper.pdf.

[245] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Zoubin Ghahramani et al. 2014, pp. 3104–3112. URL: https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html.

[246] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308. URL: https://doi.org/10.1109/CVPR.2016.308.

[247] Bruno Taillé et al. “Let’s Stop Incorrect Comparisons in End-to-end Relation Extraction!” In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 3689–3701. DOI: 10.18653/v1/2020.emnlp-main.301. URL: https://aclanthology.org/2020.emnlp-main.301.

[248] Jizhi Tang, Yansong Feng, and Dongyan Zhao. “Learning to Update Knowledge Graphs by Reading News”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 2632–2641. DOI: 10.18653/v1/D19-1265. URL: https://aclanthology.org/D19-1265.

[249] Ruixiang Tang et al. *Does Synthetic Data Generation of LLMs Help Clinical Text Mining?* 2023. arXiv: 2303.04360 [cs.CL].

[250] Yuqing Tang et al. “Multilingual Translation from Denoising Pre-Training”. In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Online: Association for Computational Linguistics, Aug. 2021, pp. 3450–3466. DOI: 10.18653/v1/2021.findings-acl.304. URL: https://aclanthology.org/2021.findings-acl.304.

[251] Erik F. Tjong Kim Sang. “Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition”. In: *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*. 2002. URL: https://aclanthology.org/W02-2024.

[252] Hugo Touvron et al. “Llama 2: Open Foundation and Fine-Tuned Chat Models”. In: *arXiv* (2023).

[253] Hugo Touvron et al. “LLaMA: Open and Efficient Foundation Language Models”. In: *ArXiv* abs/2302.13971 (2023).

[254] Bayu Distiawan Trisedya et al. “Neural Relation Extraction for Knowledge Base Enrichment”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 229–240. DOI: 10.18653/v1/P19-1023. URL: https://aclanthology.org/P19-1023.

[255] Roy Tromble and Jason Eisner. “A fast finite-state relaxation method for enforcing global constraints on sequence decoding”. In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. New York City, USA: Association for Computational Linguistics, June 2006, pp. 423–430. URL: https://aclanthology.org/N06-1054.

[256] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[257] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[258] Veniamin Veselovsky et al. *Generating Faithful Synthetic Data with Large Language Models: A Case Study in Computational Social Science*. 2023. arXiv: 2305.15041 [cs.CL].

[259] Denny Vrandečić. "Wikidata: A New Platform for Collaborative Data Collection". In: *Proceedings of the 21st International Conference on World Wide Web*. WWW '12 Companion. Lyon, France: Association for Computing Machinery, 2012, pp. 1063–1064. ISBN: 9781450312301. DOI: 10.1145/2187980.2188242. URL: https://doi.org/10.1145/2187980.2188242.

[260] Alex Wang et al. "GLUE: A multi-task benchmark and analysis platform for natural language understanding". In: *arXiv preprint arXiv:1804.07461* (2018).

[261] William Yang Wang, Jiwei Li, and Xiaodong He. "Deep Reinforcement Learning for NLP". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 19–21. DOI: 10.18653/v1/P18-5007. URL: https://aclanthology.org/P18-5007.

[262] Xuezhi Wang et al. "Self-Consistency Improves Chain of Thought Reasoning in Language Models". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=1PL1NIMMrw.

[263] Zihao Wang et al. "Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents". In: *arXiv preprint arXiv:2302.01560* (2023).

[264] Zihao Wang et al. "JARVIS-1: Open-World Multi-task Agents with Memory-Augmented Multimodal Language Models". In: *arXiv preprint arXiv: 2311.05997* (2023).

[265] Zirui Wang et al. "Towards Zero-Label Language Learning". In: *CoRR* abs/2109.09193 (2021). arXiv: 2109.09193. URL: https://arxiv.org/abs/2109.09193.

[266] Jason Wei et al. "Chain of Thought Prompting Elicits Reasoning in Large Language Models". In: *CoRR* abs/2201.11903 (2022). arXiv: 2201.11903. URL: https://arxiv.org/abs/2201.11903.

[267] Jason Wei et al. "Chain of Thought Prompting Elicits Reasoning in Large Language Models". In: *CoRR* abs/2201.11903 (2022). arXiv: 2201.11903. URL: https://arxiv.org/abs/2201.11903.

[268] Jason Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24824–24837.

[269] Sean Welleck et al. "Generating Sequences by Learning to Self-Correct". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=hH36JeQZDaO.

[270] Yixuan Weng et al. *Large Language Models are Better Reasoners with Self-Verification*. 2023. arXiv: 2212.09561 [cs.AI].

[271] Gian Wiher, Clara Meister, and Ryan Cotterell. *On Decoding Strategies for Neural Text Generators*. 2022. DOI: 10.48550/ARXIV.2203.15721. URL: https://arxiv.org/abs/2203.15721.

[272] Brandon T Willard and Rémi Louf. "Efficient guided generation for large language models". In: *arXiv preprint arXiv:2307.09702* (2023).

[273] Sam Wiseman and Alexander M. Rush. "Sequence-to-Sequence Learning as Beam-Search Optimization". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 1296–1306. DOI: 10.18653/v1/D16-1137. URL: https://aclanthology.org/D16-1137.

[274] James F. Woodward. *Making Things Happen: A Theory of Causal Explanation*. New York: Oxford University Press, 2003.

[275] Jiayang Wu et al. *Multimodal Large Language Models: A Survey*. 2023. arXiv: 2311.13165 [cs.AI].

[276] Ledell Wu et al. "Scalable Zero-shot Entity Linking with Dense Entity Retrieval". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 6397–6407. DOI: 10.18653/v1/2020.emnlp-main.519. URL: https://aclanthology.org/2020.emnlp-main.519.

[277] Lijun Wu et al. "A Study of Reinforcement Learning for Neural Machine Translation". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 3612–3621. DOI: 10.18653/v1/D18-1397. URL: https://aclanthology.org/D18-1397.

[278] Qingyun Wu et al. "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework". In: *CoRR* abs/2308.08155 (2023). DOI: 10.48550/ARXIV.2308.08155. arXiv: 2308.08155. URL: https://doi.org/10.48550/arXiv.2308.08155.

[279] Zhiyong Wu et al. *OS-Copilot: Towards Generalist Computer Agents with Self-Improvement*. 2024. arXiv: 2402.07456 [cs.AI].

[280] Guangxuan Xiao et al. *SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models*. 2023. arXiv: 2211.10438 [cs.CL].

[281] Hengjia Xiao and Peng Wang. *LLM A*: Human in the Loop Large Language Models Enabled A* Search for Robotics*. 2023. arXiv: 2312.01797 [cs.RO].

[282] Yuxi Xie et al. *Self-Evaluation Guided Beam Search for Reasoning*. 2023. arXiv: 2305.00633 [cs.CL].

[283] Tianci Xue et al. *RCOT: Detecting and Rectifying Factual Inconsistency in Reasoning by Reversing Chain-of-Thought*. 2023. arXiv: 2305.11499 [cs.CL].

[284] Ikuya Yamada et al. “Wikipedia2Vec: An Efficient Toolkit for Learning and Visualizing the Embeddings of Words and Entities from Wikipedia”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 23–30. DOI: 10.18653/v1/2020.emnlp-demos.4. URL: https://aclanthology.org/2020.emnlp-demos.4.

[285] Yiben Yang et al. “Generative Data Augmentation for Commonsense Reasoning”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1008–1025. DOI: 10.18653/v1/2020.findings-emnlp.90. URL: https://aclanthology.org/2020.findings-emnlp.90.

[286] Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=WE_vluYUL-X.

[287] Shunyu Yao et al. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. 2023. arXiv: 2305.10601 [cs.CL].

[288] Shunyu Yao et al. “Tree of Thoughts: Deliberate Problem Solving with Large Language Models”. In: *ArXiv* abs/2305.10601 (2023).

[289] Yao Yao, Zuchao Li, and Hai Zhao. *Beyond Chain-of-Thought, Effective Graph-of-Thought Reasoning in Large Language Models*. 2023. arXiv: 2305.16582 [cs.CL].

[290] Zhewei Yao et al. *ZeroQuant-V2: Exploring Post-training Quantization in LLMs from Comprehensive Study to Low Rank Compensation*. 2023. arXiv: 2303.08302 [cs.LG].

[291] Jiacheng Ye et al. *ZeroGen: Efficient Zero-shot Learning via Dataset Generation*. 2022. DOI: 10.48550/ARXIV.2202.07922. URL: https://arxiv.org/abs/2202.07922.

[292] Chih-Kuan Yeh et al. “On Completeness-aware Concept-Based Explanations in Deep Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 20554–20565. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/ecb287ff763c169694f682af52c1f309-Paper.pdf.

[293] Ori Yoran et al. “Answering Questions by Meta-Reasoning over Multiple Chains of Thought”. In: *ArXiv* abs/2304.13007 (2023).

[294] Murong Yue et al. *Large Language Model Cascades with Mixture of Thoughts Representations for Cost-efficient Reasoning*. 2023. arXiv: 2310.03094 [cs.CL].

[295] Eric Zelikman et al. *Parsel: A (De-)compositional Framework for Algorithmic Reasoning with Language Models*. 2022. URL: https://arxiv.org/abs/2212.10561.

[296] Eric Zelikman et al. “STaR: Bootstrapping Reasoning With Reasoning”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by Sanmi Koyejo et al. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

[297] Hugh Zhang et al. “Trading Off Diversity and Quality in Natural Language Generation”. In: *Proceedings of the Workshop on Human Evaluation of NLP Systems (HumEval)*. Online: Association for Computational Linguistics, Apr. 2021, pp. 25–33. URL: https://aclanthology.org/2021.humeval-1.3.

[298] Zhuosheng Zhang et al. “Multimodal Chain-of-Thought Reasoning in Language Models”. In: *ArXiv* abs/2302.00923 (2023).

[299] Lianmin Zheng et al. “Sglang: Efficient execution of structured language model programs”. In: *arXiv preprint arXiv:2312.07104* (2023).

[300] Yixin Zhong. “A Theory of Semantic Information”. In: *Proceedings* 1.3 (2017). ISSN: 2504-3900. DOI: 10.3390/IS4SI-2017-04000. URL: https://www.mdpi.com/2504-3900/1/3/129.

[301] Denny Zhou et al. “Least-to-Most Prompting Enables Complex Reasoning in Large Language Models”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=WZH7099tgfM.

[302] Daniel Zügner et al. “Language-Agnostic Representation Learning of Source Code from Structure and Context”. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=Xh5eMZVONGF.

Martin Josifoski

martin.josifoski@epfl.ch · Webpage · GitHub · LinkedIn · Scholar

EDUCATION

09.2020 – present **École Polytechnique Fédérale de Lausanne (EPFL)**
Ph.D. in Artificial Intelligence
Advisor: Robert West

09.2017 – 05.2020 **École Polytechnique Fédérale de Lausanne (EPFL)**
M.Sc. in Data Science (GPA 5.59/6.00)
Advisor: Robert West Thesis advisor (completed at ETH Zürich): Andreas Krause

09.2014 – 08.2017 **Ss. Cyril and Methodius University, Faculty of Computer and Engineering**
B.Sc. in Computer Science (GPA 9.93/10.00; top 1%)

WORK EXPERIENCE

06.2023 – 09.2023 **Internship, Office of the Chief Scientific Officer, Microsoft Research**
Hosted by Dr. Eric Horvitz

10.2019 – 05.2020 **MS thesis/Internship, Learning & Adaptive Systems Group, ETH Zürich**
Hosted by Prof. Dr. Andreas Krause

07.2019 – 10.2019 **Internship, Facebook AI Research**
Hosted by Dr. Fabio Petroni

09.2017 – 09.2019 **Research Assistantship, Data Science laboratory, EPFL**
Supervised by Prof. Dr. Robert West

SELECTED PUBLICATIONS

Click here for the complete list

Preprint **The Era of Semantic Decoding**
*Authors: **Martin Josifoski***, Maxime Peyrard*, Robert West*

Preprint **Flows: Building Blocks of Reasoning and Collaborating AI**
*Authors: **Martin Josifoski***, Lars Klein*, Maxime Peyrard, Yifei Li, Saibo Geng, Julian Paul Schnitzler, Yuxing Yao, Jiheng Wei, Debjit Paul, Robert West*
→ A precursor to the aiFlows library, which has +200 stars on GitHub, for building structured interactions involving tools, AI systems and humans, whose development I am leading.

ICLR'24 **Evaluating Language Model Agency Through Negotiations**
*Authors: Tim Davidson, Veniamin Veselovsky, **Martin Josifoski**, Maxime Peyrard, Antoine Bosselut, Michal Kosinski, Robert West*

JMLR **Scalable PAC-Bayesian Meta-Learning via the PAC-Optimal Hyper-Posterior: From Theory to Practice**
*Authors: Jonas Rothfuss, **Martin Josifoski**, Vincent Fortuin, Andreas Krause*

EMNLP '23 **Exploiting Asymmetry for Synthetic Training Data Generation: SynthIE and the Case of Information Extraction**
*Authors: **Martin Josifoski**, Marija Sakota, Maxime Peyrard, Robert West*

EMNLP '23 **Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning**
*Authors: Saibo Geng, **Martin Josifoski**, Maxime Peyrard, Robert West*

EACL '23 **Language Model Decoding as Likelihood-Utility Alignment**
*Authors: **Martin Josifoski**, Maxime Peyrard, Frano Rajic, Jiheng Wei, Debjit Paul, Valentin Hartmann, Barun Patra, Vishrav Chaudhary, Emre Kıcıman, Boi Faltings, Robert West*

EMNLP '22 **Invariant Language Modeling**
*Authors: Maxime Peyrard, Sarvjeet Singh Ghotra, **Martin Josifoski**, Vidhan Agarwal, Barun Patra, Dean Carignan, Emre Kiciman, Robert West*

NAACL '22 **GenIE: Generative Information Extraction**
*Authors: **Martin Josifoski**, Nicola De Cao, Maxime Peyrard, Fabio Petroni, Robert West*

ICML '21 **PACOH: Bayes-Optimal Meta-Learning with PAC-Guarantees**
*Authors: Jonas Rothfuss, Vincent Fortuin, **Martin Josifoski**, Andreas Krause*

EMNLP '20 **Zero-shot Entity Linking with Dense Entity Retrieval**

*Authors: Ledell Wu, Fabio Petroni, **Martin Josifoski**, Sebastian Riedel, Luke Zettlemoyer*
→ The project's repository, open sourced at the end of my internship, has +1,000 stars.

WSDM '19 **Crosslingual Document Embedding as Reduced-Rank Ridge Regression**

*Authors: **Martin Josifoski**, Ivan Paskov, Hristo Paskov, Martin Jaggi, Robert West*

FELLOWSHIPS & ACHIEVEMENTS

2023 - present ***Microsoft's Accelerating Language Model Research in Academia Program*** (US$50,000)
Title: "Effective and Transparent Collaboration Among LLM Agents"

2022 – present ***PhD Fellowship*** from the Swiss Data Science Center (max 3 years; CHF175,000)
Title: "Better Decoding Algorithms for Large Language Models"

2022 – 2023 ***Google's Research Collab Grant*** (collaboration + US$100,000)
Title: "Plan-Before-You-Implement: A Higher-Level Reasoning Paradigm for LMs of Code"
Google sponsor: Michele Catasta

2020 – 2023 ***Microsoft's Turing Academic Program*** (collaboration + compute)
Microsoft collaborators: Emre Kıcıman, Jason Eisner and members from the MS-Turing team (Vishrav Chaudhary, Vidhan Agarwal, Barun Patra et al.)

2020 – 2021 ***PhD Fellowship*** from EPFL for the first year of the PhD (CHF55,000)

2017 – 2019 ***M.Sc. Research Scholarship*** at EPFL (CHF38,000)

2015 – 2017 National ***Scholarship*** for university students with outstanding academic achievements (EUR3,000)

2010 – 2014 National ***Scholarship*** for high school students with outstanding academic achievements (EUR1,500)

2010 – 2014 High-School Tuition ***Scholarship*** based on entrance exam results (EUR10,000)

HONORS & AWARDS

2015 ACM ICPC Southeastern Europe Regional contest

2014 Bronze at National Informatics Olympiad

2013 International Mathematical Olympiad participation

2013 Balkan Mathematical Olympiad participation

2013 Gold Medal at National Mathematical Olympiad

2011 Bronze medal at Junior Balkan Mathematical Olympiad

PROGRAMMING SKILLS

Python, PyTorch, TensorFlow, Spark, Java, SQL, C++. HTML, JavaScript

TALKS

2024 Learn to Develop and Customize AI Workflows with Flows
- **Workshop at Applied Machine Learning Days (AMLD)**

2023 Flows: Building Blocks of Reasoning and Collaborating AI
- **Google Research**
- **Berkeley**
- **NEC Lab**

2023 Symbolic Intermediate Representations for Reasoning and Collaborating AI
- **Swiss Data Science Center**

2022 GenIE: Generative Information Extraction
- **NEC Lab**
- **Wikimedia Research**

2022 Exploiting Asymmetry for Synthetic Training Data Generation: SynthIE and the Case of Information Extraction, **EPFL Open House**

2019 Crosslingual Document Embedding as Reduced-Rank Ridge Regression
- **Wikimedia Research**
- **Wikimania (Wikipedia annual conference)**