Toward Code Generation: A Survey and Lessons from Semantic Parsing

Celine Lee
Intel Labs, University of Pennsylvania
celine.lee@intel.com

Justin Gottschlich
Intel Labs, University of Pennsylvania
justin.gottschlich@intel.com

Dan Roth
University of Pennsylvania
danroth@seas.upenn.edu

Abstract
With the growth of natural language processing techniques and demand for improved software engineering efficiency, there is an emerging interest in translating intention from human languages to programming languages. In this survey paper, we attempt to provide an overview of the growing body of research in this space. We begin by reviewing natural language semantic parsing techniques and draw parallels with program synthesis efforts. We then consider semantic parsing works from an evolutionary perspective, with specific analyses on neuro-symbolic methods, architecture, and supervision. We then analyze advancements in frameworks for semantic parsing for code generation. In closing, we present what we believe are some of the emerging open challenges in this domain.

1 Introduction

Machine programming (MP) is the field concerned with the automation of all aspects of software development [28]. According to Gottschlich et al. [28], the field can be reasoned about across three pillars: intention, invention, and adaptation (Figure 1). Intention is concerned with capturing user intent, whether by natural language, visual diagram, software code, or other techniques. Invention explores ways to construct higher-order programs through the composition of existing – or novel creation of – algorithms and data structures. Adaptation focuses on transforming higher-order program representations or legacy software programs to achieve certain characteristics (e.g., performance, security, correctness, etc.) for the desired software and hardware ecosystem.

In this paper, we discuss (i) semantic parsing and (ii) code generation, which chiefly fall within the boundaries of the intention and invention pillars, respectively. In the context of natural language (NL), semantic parsing is generally concerned with converting NL utterances (i.e., the smallest unit of speech) to structured logical forms. Once done, these logical forms can then be utilized to perform various tasks such as question answering [41, 42, 47, 80, 86], machine translation [5, 81], or code generation [38, 49, 73, 87], amongst other things. For semantic parsing for code generation, the end logical form will usually be some form of software code. This may be the direct output program [41, 49, 90, 94] or an intermediate representation of code [21, 42, 68, 87, 88], which can subsequently be fed into a program synthesis component to generate syntactically sound and functionally correct code.

In this survey paper, we first provide an overview of the processes of NL semantic parsing and of code generation in Sections 2 and 3, respectively. In Section 4, we summarize the evolution of techniques in NL and code semantic parsing from the 1970s onward. Section 5 explores the question of supervision for semantic parsing tasks, while Section 6 discusses modern advances in neural semantic parsing for code generation. To conclude, we consider some possible future directions of semantic parsing for code generation.

2 Natural Language Semantic Parsing

Natural language analysis can be segmented into at least two fundamental categories: syntax and semantics. In general, syntax is the way a natural language phrase is structured; the order and arrangement of words and punctuation. The semantics is the meaning that can be derived from such syntax. For example, the two sentences “the boy cannot go” and “it is not possible for the boy to go”, are semantically equivalent even though they are syntactically different. By lifting semantics from syntax, NL semantic parsers can map semantically equivalent sentences to the same logical form, even when they have different syntaxes. An example of this is shown in Figure 2, where two syntactically different sentences are shown to be semantically equivalent using an abstract meaning representation (AMR) as its logical form (more details in Section 2.4). For the remainder of this section, we discuss how NL semantic parsers extract meaning from natural language utterances into various machine-understandable representations.
Abstract meaning representation (AMR) of two semantically equivalent, but syntactically different sentences (credit: Banarescu et al. [12]).

### 2.1 Purpose of Natural Language Semantic Parsing

The general purpose of structured logical forms for natural language is to have a representation that enables natural language understanding (NLU) and automated reasoning, amongst other things. Historically, NL semantic parsers have enabled computers to query databases [89], play card games [27], and even act as conversational agents such as Apple’s Siri and Amazon’s Alexa. Semantic parsing has taken on many forms. Early hand-crafted rules made way for supervised learning techniques. Then, due to the labor-intensive and error-prone process of hand-labeling data for paired natural language and logical form datasets, researchers turned their attention toward weakly supervised [9, 20] and other learning techniques that tend to require less labeled data.

### 2.2 Components of a Semantic Parsing System

Given an NL input, a NL semantic parsing system generates its semantic representation as a structured logical form. These logical forms are also known as meaning representations or programs. To generate these outputs, semantic parsing pipelines usually trained by some learning algorithm to produce a distribution over the output search space and then search for a best scoring logical form in that distribution. This logical form can then be executed against some environment to carry out an intended task (e.g. to query a database). Because these logical forms are generally designed to be machine-understandable representations, they tend to have a structured nature, following an underlying formalism by which some grammar can be used to derive valid logical forms. Insight into the grammar of a set of logical forms can be leveraged by different semantic parsing methods to improve performance. For example, semantic parsing for code generation should have a meaning representation that can deterministically be transformed into valid code. Therefore, the output space can be constrained by the underlying grammar formalisms that define syntactically sound code.

### 2.3 Grammars

Grammar in the context of natural language processing is a set of rules that govern the formation of some structure (parsing of NL utterances) to ensure well-formedness. A familiar example is English grammar: number, tense, gender and other features must be consistent in an English sentence.

The sentence “he eat apples” is grammatically incorrect because a single noun acts with a verb that must end in ‘s.’ Likewise, in semantic parsing, a logical form can be grammatically constrained to ensure well-formedness. A set of grammatical rules constrains the search space of possible outputs. Combinatory categorical grammar (CCG) [76, 77] is one such example of a popular grammar formalism. It has historically made frequent appearances in semantic parsing models [45, 91, 92] due to its ability to jointly capture syntactic and semantic information of the constituents which it label. Consider the example in Figure 3. By pairing syntactic and semantic information, CCGs can describe textual input for a machine to understand the semantics with the syntax.

### 2.4 Meaning Representations

Meaning representations are the end product of NL semantic parsing. The output logical formula, or meaning representation, grounds the semantics of the natural language input in the target machine-understandable domain. A simple example of logical forms is a database query. Popular datasets explored in semantic parsing include Geo880, Jobs640, ATIS, and SQL itself [20, 37, 89]. An example of a SQL query is shown in Figure 4. Another class of semantic parsing outputs is NL instructions to actions, such as the model in Figure 5 for learning Freecell card game moves. Note that in these cases, the semantic parser is specific to a particular domain, and may not generalize to out-of-domain inquiries or differing data structures.

NL semantic parsing efforts have also translated natural language input semantics into lambda calculus expressions. The expressive power of lambda calculus allows the incorporation of constants, logical connectors, quantifications, and lambda expressions to represent functions [92].

Sentence: what countries border France
Logical Form: $\lambda x.\text{country}(x) \land \text{borders}(x, \text{France})$
Meaning representations can also take on graph-based forms. Graph-based formalisms are advantageous because they are easier for humans to read and they are abstracted from the syntax of the input natural language sentences. Popular graphical representations are the Abstract Meaning Representation [12], Semantic Dependency Parsing [17, 59], Universal Conceptual Cognitive Annotation [1], Dependency-based Compositional Semantics [47], and CCGBank [34].

While most general purpose programming languages are not necessarily abstract in their representations of intention (source code often involves details about variables, memory allocation, class structure, etc.), code can serve as one of the more fundamental machine-understandable logical forms. To lift semantics from the code’s syntax, recent works have gone into developing and using meaning representations of code, such as the abstract syntax tree (AST) [2–4, 67, 71, 72], sketching [74], simplified parse tree (SPT) [51], contextual flow graph (XFG) [13], context-aware semantic structure (CASS) [85], and program-derived semantics graph (PSG) [35]. While none of these representation structures are identical, they somewhat share the common goal of being a vehicle of machine-understandable meaning of the code, abstracted to some extent from the syntax of code. We discuss the application of some of these structures in Section 3.

3 Program Synthesis / Code Generation

In this section, we review a subset of existing efforts for program synthesis and code generation. Program synthesis is the task of generating software from some high-level program specification [54]. The purpose of this section is to highlight parallels between this process of using a high-level program specification to generate code, and semantic parsing’s translation of natural language into meaning representations. Insights from the fields of program synthesis and machine programming can guide semantic parsing efforts, and vice versa.

Code has many different dimensions. Despite what could be construed as a strict syntax for compile-ability, source code can be reasoned about on many different levels: semantics, runtime, memory footprint, and compiler details, to name a few. As a result, design and selection of a code representation structure, and the subsequent reasoning about said structure is an interesting and nuanced task.

Many code recommendation systems use code similarity to retrieve different implementations of the same intention. Semantic code representations can provide a foundation on which to extract code similarity. Facebook et al.’s Aroma system for code recommendation [51] represents a program by a simplified parse tree (SPT): the parse tree is defined as a recursive structure of keyword and non-keyword tokens, agnostic to implementation programming language. Intel et al.’s MISIM system for code similarity [85] defines the context-aware semantic structure (CASS), which leverages the structural advantages of the SPT, but also carries the advantage of incorporating customizable parameters for a wider variety of code context scenarios. The CASS enables configuration at different levels of granularity, allowing it to be more flexible for general, rather than domain-specific, usage. Iyer et al.’s [35] present the program-derived semantics graph (PSG) to capture semantics of code at multiple levels of abstraction. Additionally, it offers one layer of concrete syntactic abstraction that is programming language specific, which allows it to be converted into source code.

Program synthesis has also come in the flavor of machine translation [18, 67, 81], pseudocode-to-code transformation [44], and inductive program synthesis by input-output examples [11, 31, 57]. While machine translation found its infancy in translation between human languages, a series of works has gone into translating from natural language to programming languages. Wong and Mooney’s [81] integrate machine translation techniques with semantic parsing to create a syntax-based WASP: word alignment-based semantic parsing. Quirk et al. [67] further combine machine translation to extract ASTs consistent with a constructed grammar for If-This-Then-That (IFTTT) recipes. Chen et al. [18] tackle program translation by employing tree-to-tree deep neural networks to migrate code from one language into an ecosystem built in a different language. Kulal et al. [44] perform the task of mapping line-to-line pseudocode to code for program synthesis. Inductive program synthesis allows us to produce a program consistent with a set of input-output examples [11, 24, 30, 31, 48, 57, 63, 66]. In these approaches, the program synthesis problem can be framed as the following: first, encode input-output examples as constraints; then, search over the space of possible programs with a constraint solver. A key challenge in this technique is scalability. Pu et al. [66] demonstrate that inductive program synthesis can be made a scalable task by having a model iteratively select a small subset of representative examples, then train on each small subset at a time. Another aspect to consider is evaluation. A fitness function determines how well a given candidate program satisfies some set of requirements. Given these requirements as a set of input-output examples, such as in inductive program synthesis, the fitness function can be learned [53], resulting in more efficient program synthesis because it is biased to search likely programs.
The task of program synthesis tends to be more tractable if the output programming language is less complex and thus the output space is more compact. Declarative programming languages such as SQL [15], Halide [69], and GraphIT [93] express the intention of a computation without detailing its execution. For this reason, these types of languages are also called intentional programming languages in the field of machine programming. As a function of the syntax and semantics being closely related, programs written in intentional programming languages tend to be easier to reason about in a semantic space. A byproduct of this phenomenon is that intentional programming languages can also be easier to optimize. Neo [56] and Bao [55] are learned neural models that optimize SQL queries. Verified lifting [40] also demonstrates an ability to optimize Halide code.

As we discuss the evolution of semantic parsing in the next few sections, we identify parallels between these techniques used for program synthesis techniques and the techniques used in semantic parsing.

4 Evolution of Semantic Parsing

This section will review the evolution of NL semantic parsing techniques from early hand crafted rules in the 1970s to the rise of learned grammars via statistical learning methods in the late 1990s and early 2000s, to modern neural semantic parsing efforts.

4.1 Rules-based Techniques for Semantic Parsing

Many early approaches to NL semantic parsing used hand-crafted syntactic pattern and phrase matching to parse the meaning from a given natural language input [39, 82]. These rules were then augmented by grammars that incorporate knowledge of semantic categories [33, 50, 80] so that similar words can be generalized to sets of defined categories and then constrain the output space of the logical forms. However, as a function of being based off of pattern-matching rules, these systems are brittle, and as a function of relying on defined semantic categories, these grammar-constrained systems are domain-specific.

4.2 Grammar and Rule Induction for Semantic Parsing

Statistical methods for the induction of NL semantic parsing grammars rose in the late 1990s and early 2000s. Zelle and Mooney [90] present a system that learns rules for language parsing instead of relying on hand-crafting them. The system employs inductive logic programming methods to learn shift-reduce parsers that map sentences into database queries. Kate et al. [41] also induce transformation rules that map natural language to logical form. By learning and recursively applying these rules, Kate et al.’s system maps natural language inputs (or their corresponding syntactic parse trees) to a parse tree in the target representation language.

4.3 Neural Models for NL Semantic Parsing

The modern rise in neural models has been marked by the increase in papers exploring encoder-decoder sequence-to-sequence (seq2seq) frameworks for semantic parsing. These encoders and decoders are often built with recurrent architectures such as LSTM cells (see Figure 6) in an effort to better capture the dynamic and sequential nature of both natural and programming languages. Neural models can enable systems to automatically induce grammars, templates, and features instead of relying on humans to manually annotate them. This provides the model with flexibility to generalize across languages and representations [42]. These encoder-decoder models can bear a resemblance to machine translation systems [81] except that NL semantic parsing systems produce a structured logical form, while machine translation systems traditionally translate a natural language input into another unstructured natural language output [5]. The formal output structure of semantic parsing enables neural
decoders to lean on the logic of the induced grammars, while retaining the simplicity of neural translation without latent syntactic variables\cite{42, 68, 87}. Thus, this type of modeling can generalize to different environments while reducing demand for custom-annotated datasets. It should be noted, however, that these neural models tend to have a weakened ability to leverage the logical compositionality of traditional rules-based approaches.

Dong and Lapata \cite{21} present a NL semantic parsing model that employs an attention-enhanced encoder-decoder. In a nod to the hierarchical structure of their system’s logical forms, they design a top-down tree decoder to more faithfully decode well-formed meaning representations. Decoding of each token is augmented by: (i) soft alignments from attention computed by the encoder and (ii) parent-feeding connections to provide contextual information from prior time steps. The resulting model has competitive performance and increased flexibility without need for any human-engineered features, making it more scalable to different domains. This work marks the advent of growing attention to neural models for semantic parsing.

A more in-depth discussion of modern techniques in neural semantic parsing models is presented in Section 6, where we focus on semantic parsing models for code generation.

5 Supervision in Semantic Parsing

Parallel to the evolution of NL semantic parsing techniques is the evolution of supervision for semantic parsing. Fully-supervised learning customarily mandates a corpus of paired natural language utterances and their corresponding logical forms. However, this can be expensive to generate and difficult to scale.

Supervision by Denotation. Supervision by denotation is proposed as an alternative, more scalable approach to NL semantic parsing. Instead of directly evaluating the success of a semantic parser by comparing the output to a gold standard “correct” logical form as done in fully-supervised learning, NL semantic parsing can be driven by the response of the context or environment. Clarke et al. \cite{19} present a learning algorithm that relies only on a binary feedback signal for validation. The resulting parser is competitive with fully supervised systems in the Geoquery domain, demonstrating the feasibility of supervision by denotation. Liang et al. \cite{47} also elide the need for labeled logical forms by inducing a NL semantic parsing model from question-answer pairs. They introduce a new semantic representation, the dependency-based compositional semantics (DCS) to represent formal semantics as a tree. The model maps a NL utterance to DCS trees, aligns predicates within each tree to the span in the utterance that triggered it, and beam search for the best candidates.

However, at least two primary challenges arise with supervision by denotation. (1) The space of potential denotations may be large. This can increase computational tractability challenges in the search problem for intermediate latent representations. (2) A weaker response signal has the potential for spurious logical forms: semantic parsing outputs that happen to produce the correct response, but do not actually carry the correct semantics. Modifications to supervision by denotation have been proposed to address these challenges. Goldwasser and Roth \cite{27} present a method of learning from natural language instructions and a binary feedback signal to address these issues. By capturing alignments between NL instruction textual fragments and meaning representation logical fragments, the model can more closely interpret NL instructions and hopefully reduce the potential for spurious forms. Pasupat and Liang \cite{62} propose a system that performs dynamic programming on denotations to address the issue of exploding latent search space and spurious results.

By observing that the space of possible denotations grows more slowly than the space of logical forms, this dynamic programming approach can find all consistent logical forms up to some bounded size. Then, because spurious logical forms will generally give a wrong answer once the context changes, they generate a slightly modified context and examine whether the logical form produces the correct denotation in this alternate context. If it does not, this spurious logical form can be discarded. The combination of limited search space and pruning of spurious forms allows them to use denotation as a stronger supervision signal.

Weakly Supervised Learning. Artzi and Zettlemoyer \cite{8} use conversational feedback from un-annotated conversation logs as a form of weak supervision to induce a NL semantic parser. In this approach, Artzi and Zettlemoyer model the meaning of user statements in the conversational log with latent variables, and define loss as a rough measure of how well a candidate meaning representation matches the expectations about the conversation context and dialog domain. Artzi and Zettlemoyer \cite{9} demonstrate the viability of weak supervision for mapping natural language instructions to actions in the robotics domain. They run the output of a parsing component through an execution component and observe the result relative to a binary-feedback validation function in the environment.\footnote{We discuss other emerging works of weak supervision, such as Snorkel \cite{70}, in Section 7.}

Self-supervised Learning. Poon and Domingos \cite{65} present a self-supervised system for NL semantic parsing by recursively clustering semantically equivalent logical forms to abstract away syntactical implementation details. This method, however, does not ground the induced clusters in an ontology. Poon \cite{64} subsequently proposes a grounded unsupervised learning approach that takes a set of natural language questions and a database, learns a probabilistic semantic grammar over it, then constrains the search space using the database...
schema. For a given input sentence, this system annotates the sentence’s dependency tree with latent semantic states derived from the database. The dependency-based logical form is additionally augmented with a state space to account for semantic relationships not represented explicitly in the dependency tree. This semantically-annotated tree can then be deterministically converted into the logical form, without ever needing pre-labeled data. Goldwasser et al. [26] suggest a confidence-driven self-supervised protocol for NL semantic parsing. The model compensates for the lack of training data by employing a self-training protocol driven by confidence estimation to iteratively re-train on high-confidence predicted datapoints.

Open vocabulary NL semantic parsing replaces a formal knowledge base with a probabilistic base learned from a text corpus. In this framework, language is mapped onto queries with predicates (parts of the structure that make claims about the actions or characteristics of the subject) derived directly from the domain text instead of provided through a grammar. The model must also learn the execution models for these induced predicates [43, 46]. Gardner and Krishnamurthy [25] address the challenge of generalizing NL semantic parsing models to out-of-domain and open-vocabulary applications. Instead of mapping language to a single knowledge base query like traditional NL semantic parsers, this approach maps language to a weighted combination of queries plus a distributional component to represent a much broader class of concepts. This approach can expand the applicability of semantic parsing techniques to more complex domains.

6 Modern Advances in Neural Semantic Parsing for Code Generation

Recent semantic parsing efforts have largely centered around neural models, especially encoder-decoder networks. In this section, we discuss modern advances in neural models for semantic parsing for code generation.

6.1 Context and Variable Copying

Information to generate a NL semantic parse is not always encapsulated entirely in a single input sentence. Surrounding context can provide valuable information for disambiguation in NL semantic parsing [8]. Context can be fed into the encoder to incorporate context to encoder-decoder models.

Iyer et al. [38] develop an architecture that leverages programmatic context to write contextually-relevant code. This encoder-decoder system feeds the encoder the input utterance along with featureized representations of environment identifiers, such as type and name of each variable and method. The decoding process attends first to the natural language, then to the variables and methods of the environment. Performing this attention in two steps can help the model match words in the natural language to the environment identifier representations. Additionally, a supervised copy mechanism [29] based on attention weights is presented to copy an environment token to the output. This allows the model to copy in variable names not seen in the training data or in the input utterance.

6.2 Grammar in Neural Semantic Parsing Models

Grammar also can guide and constrain the decoding process of neural encoder-decoder models. Xiao et al. [83] propose a sequence-based approach to generate a knowledge base query: they sequentialize the target logical forms, then demonstrate the advantage of incorporating grammatical constraints into the derivation sequence. Further, when querying information from semi-structured tables, Krishnamurthy et al. [42] demonstrate the benefits of enforcing type constraints during query generation. They show that including explicit entity linking during encoding can bolster accuracy because it ensures the generation of well-typed logical forms.

As discussed in Section 3, code is dynamic in its representation and semantics, making grammar especially interesting for generating source code. Unlike prior works which focus on domain-specific languages, which may have limited scope and complexity, Yin and Neubig [87] and Ling et al. [49] present methods for generating high-level general-purpose programming language code such as Python. Ling et al. [49] present a data-driven sequence-to-sequence code generation method that implements multiple predictors: a generative model for code generation, and multiple pointer models to copy keywords from the input. Building off of Ling et al. [49], Yin and Neubig [87] point out that general purpose languages are generally more complex in schema and syntax than other domain-specific semantic parsing challenges. Therefore, code generation models could benefit from a grammar that explicitly captures the target programming language syntax as prior knowledge. Yin and Neubig [87] use abstract syntax trees (ASTs) to constrain the search space by exclusively allowing well-formed outputs: the model generates a series of actions that produce a valid AST. The actions themselves are derived from the underlying syntax of the programming language. This approach demonstrates improved Python code generation, but Yin and Neubig point out a decline in performance as the AST becomes more complex, noting that there may be space for improvement in scalability.

Code idioms are bits of code that occur frequently across a code repository and tend to have a specific semantic purpose. They can be used to enhance grammars in semantic parsing. Iyer et al. [36] apply idiom-based decoding to reduce the burden on training data in grammar-constrained semantic parsing systems. The top-K most frequently seen idioms in the training set are used to collapse the grammar for decoding. This compact set of rules helps supervise the learning of the semantic parser. However, there is a limit to how much idioms help with performance. The paper notes that adding
too many idioms may cause the model to over-fit and become less generalizable. Shin et al. [73] mine code idioms with the objective of helping models reason about high-level and low-level reasoning at the same time. The mined code idioms are incorporated into the grammar as new named operators, allowing the model to emit entire idioms at once. Incorporating the code idioms into the vocabulary of the neural synthesizer enables it to generate high-level and low-level program constructs interchangeably at each step, which is more like how human programmers write code.

6.3 Sketching

Solar-Lezama et al. [75] present the concept of sketching to generate programs from a high-level description. Through sketching, developers communicate insight through a partial program, i.e. the sketch, that describes the high-level structure of an implementation. Then, synthesis procedures can generate the full implementation of this sketch in programming language.

Dong and Lapata [22] present a semantic parsing decoder that operates at varying levels of abstraction for encoder-decoder semantic parsing. By operating as a representation of the basic meaning, the sketch can provide a basic idea of the meaning of the utterance while abstracting away from the syntactic details of the actual implementation. The decoder presented in this paper operates in two stages: (1) generating a coarse sketch of the meaning representation that omits low-level details, and (2) filling in those missing details of arguments and variable names. This coarse-to-fine decoding has the advantage of disentangling high and low-level semantic information so that the program can be modeled at different granularities. An attention-enhanced RNN serves as the fine meaning decoder, using the coarse sketch to constrain the decoding output. This approach does not utilize syntax or grammatical information because the sketch naturally constrains the output space.

Nye et al. [58] present a method to dynamically integrate natural language instructions and examples to infer program sketches. Previous systems have mostly used static, hand-designed intermediate sketch grammars, which tend to be rigid in how much a system relies on pattern recognition vs. symbolic search. When given an easy or familiar task, Nye et al. [58]’s neuro-symbolic synthesis system can rely on learned pattern recognition to produce a more complete output. When given a hard task, the system can produce a less complete sketch and spend more time filling in that sketch using search techniques. By learning how complete to make the sketch for different applications, this model can employ sketching in a more fitting and customized manner.

6.4 Conversational Semantic Parsing

A remaining challenge in neural semantic parsers is interpretability. Dong et al. [23] outline a confidence modeling framework that characterizes three uncertainties: model uncertainty, data uncertainty, and input uncertainty. By computing these confidence metrics for a given prediction and using them as features in a regression model, they obtain a confidence score which can be used to identify which part of the input or model contributes to uncertain predictions. This uncertainty modeling allows models to better understand what it does and does not know. The ability to target weak parts of a model leads us to conversational programming approaches, where the model can query the user for missing or uncertain information.

When performing NL semantic parsing to extract some logical flow, conversational AI allows models to remedy incomplete or invalid requests. By engaging in a back-and-forth dialogue between the program and the human user, gaps and errors in logical forms can be filled and remedied, respectively. Additionally, clarifications can reduce the search space to promise a more accurate output [8, 16, 52].

7 Future Directions

The task of connecting human communication with machine data and capabilities, as in the intention pillar of machine programming, has seen considerable headway over the past few decades. Moving forward in semantic parsing, we identify four areas of consideration for the next steps of semantic parsing for code generation:

Evaluation. Current methods of evaluation for semantic parsing correctness use strict matches to gold standards, denotation, hand-wavy heuristics, or some other imperfect form of measuring understanding of semantic meaning. Exact match evaluation may incorrectly penalize mismatching logical forms that actually produce the same semantic meaning. Supervision by denotation can incorrectly reward spurious logical forms. Hand-wavy heuristics such as Bleu score [61], which evaluates the “quality” of machine translation outputs via a modified n-gram precision metric, have been shown to not reliably capture semantic equivalence for natural language [14] or for code [79]. Efforts to resolve the weaknesses of each of these evaluation methods [20, 27, 62, 64, 65] have not been ineffective, but do not seem to fully address the key issue, which is that an evaluator for semantic parsing should have an understanding of semantic reasoning. We propose that an important next step may be to research either or both of: (i) an evaluator that can identify matching core semantics of different logical forms, and (ii) a meaning representation that has one unique representation for any given semantic intention.

Evaluation of a model’s actual understanding could be another open challenge in semantic parsing. Rules- and grammar-based systems [10, 33, 39, 41, 50, 80, 90] are generally more transparent and thus more interpretable than neural models because the rules on which they are built.
We recommend research moving forward to consider how to well-known and novel extensions to the seq2seq model im-
This suggests a need to make more drastic changes to the problem more feasible. This can be done either by work-
representation could then be used to translate this refined translated from source code into natural language. Then
language, as discussed in this paper, it could also then be
by conversation. ” Not only could software be generated from
meaning representation, and code, this unified inter-
be examined as the logic of the model. However, neu-
ral models tend to function more as “black boxes,” where induced logic is mostly evaluated by examining the outputs.
We recommend research moving forward to consider how to design this neural learning to better capture the complex na-
ture of both natural language and source code in translating natural language into source code. Compositional general-
ization is one way to evaluate how well a model understands natural language. Oren et al. [60] investigate compositional
generalization in semantic parsing. They find that while well-known and novel extensions to the seq2seq model im-
prove generalization, performance on in-distribution data is still significantly higher than on out-of-distribution data.
This suggests a need to make more drastic changes to the neural encoder-decoder approach in order to inject funda-
mental “understanding.” Neural Module Networks (NMNs) can also provide interpretability, compositionality, and im-
proved generalizability to semantic parsers [32, 78]. Gupta et al. [32] investigate how multiple task-specialized NMNs can be composed together to perform complex reasoning over unstructured knowledge such as natural language text. However, the advanced interpretability of NMNs comes at the cost of restricted expressivity because each NMN needs to be individually defined. Gupta et al. [32] reflect that this tradeoff suggests future further research to bridge these rea-
soning gaps.

**Code Semantics Representation.** We perceive that an ongoing challenge in semantic reasoning about language and code may be the lack of a unified code representation that can capture semantics and details enough to resolve am-
biguity but remain agnostic of programming language. The graph-based meaning representations discussed in Section
2.4 are promising avenues to explore. They have shown to be effective in code retrieval, so we propose they be deployed in a semantic parsing framework to evaluate their viability for semantic parsing and code generation.

Big picture, developing this language-agnostic meaning representation is a promising direction to facilitate a full machine programming pipeline. If the meaning representation is transformable in all the directions between natural language, meaning representation, and code, this unified inter-
mediate representation enables a sort of “programming by conversation.” Not only could software be generated from
language, as discussed in this paper, it could also then be translated from source code into natural language. Then
once in natural language form, intention could be iterated upon via an interactive conversational agent. The iterated
representation could then be used to translate this refined intention back into software.

**Scope.** Most current semantic parsing models reduce the scope of the problem (and thus, often, output space) to make the problem more feasible. This can be done either by working with DSLs, using a selected subset of APIs, or by working only with code snippets of a limited length and complexity. However, actual functional programs tend to be longer and more complex than the code snippet outputs in current se-
monic parsing models. Generation of programs in a GPL [49] is generally a more complex task than in a DSL due to the wide variety of domains and the high-dimensional code reasoning space. Eventually, we want to be able to work dyn-
amically with larger programs. Steps toward incorporating context to and using conversational AI in neural semantic parsing models have shown promising results, and should continue to be pursued.

As the field moves into working with larger programs, existing datasets that match natural language utterances to code snippets will likely not be enough. To obtain more data for neural deep learning, we recommend using data generation models such as Snorkel [70]. Snorkel employs weak supervision to create training data without any need for hand-labeled data. We also suggest that new datasets be developed that account for longer chunks of code. This can be done by factoring in context variables and functions, or by mining larger chunks of code with periodic natural language annotations in the form of comments.

**User Interfacing.** Future work could also consider the actual impact of code generation on programmer productivity. Xu et al. [84] conducted a study that found no impact on code correctness or generation efficiency when the subjects were asked to work in a programming environment IDE with a code generation plug-in. This begs the question of how to develop future models to be not just user-friendly, but user aiding. Additionally, user interfaces for code generation may not share the same vocabulary between trained programmers and non-technical developers. There may be room for natural language understanding research advances for models to intuit programming concepts from plain English, free of such jargon as "list," "graph," and "iterate."

8 Broader Impact
The objective of this paper is to compile information and synthesize ideas about the fields of semantic parsing for natural language and for code. Since the basis of this paper is primarily educational, we see its broader impacts to largely be positive.

Semantic parsing for code generation will likely grant more people a greater capacity to create programs. How-
ever, this enhanced capability also carries potential negative impacts. One possible misuse is the creation of malicious code. The models outlined in this paper could be trained to learn to write computer viruses, identify security breaches in other peoples’ networks, or any of a number of dangerous behaviors. Aside from continuing to make this malicious code illegal, it may be worthwhile to embed safety features into the prior knowledge of these models, to discourage de-
structive code. On the flip side, models for code generation
could also in the worst case generate programs with egregiously poor performance. If deployed, these programs will consume excessive computation and energy, thus negatively impacting the environment. One way to mitigate this harm is to encourage code reviews for generated code, just as code reviews exist now for human-created code. Another potential misuse of this technology that we would be concerned about is computer science students using this technology to complete their programming assignments, rather than learning how to write them themselves. This will be difficult to discourage from the model development side, but development and usage of a technology to identify coding style could help instructors detect whether students are writing their own code. Overall, researchers, as designers of these semantic parsing systems, should consider the safety and impact of their work on the many different realms of society.

References

[1] Omri Abend and Ari Rappoport. 2013. Universal Conceptual Cognitive Annotation (UCCA). In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Sofia, Bulgaria, 228–238. https://www.aclweb.org/anthology/P13-1023

[2] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to Represent Programs with Graphs. In International Conference on Learning Representations.

[3] Uri Alon, Omer Levy, and Eran Yahav. 2019. code2seq: Generating Sequences from Structured Representations of Code. In International Conference on Learning Representations.

[4] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. Code2vec: Learning Distributed Representations of Code. Proc. ACM Program. Lang. 3, POPL, Article 40 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290353

[5] Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic Parsing as Machine Translation. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Association for Computational Linguistics, Sofia, Bulgaria, 47–52. https://www.aclweb.org/anthology/P13-2009

[6] Yoav Artzi, Dipanjan Das, and Slav Petrov. 2014. Learning Compact Lexicons for CCG Semantic Parsing. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Doha, Qatar, 1273–1283. https://doi.org/10.3115/v1/D14-1134

[7] Yoav Artzi, Nicholas Fitzgerald, and Luke Zettlemoyer. 2014. Semantic Parsing with Combinatory Categorial Grammars. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts. Association for Computational Linguistics, Doha, Qatar. https://www.aclweb.org/anthology/D14-2003

[8] Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping Semantic Parsers from Conversations. In Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Edinburgh, Scotland, UK., 421–432. https://www.aclweb.org/anthology/D11-1039

[9] Yoav Artzi and Luke Zettlemoyer. 2013. Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions. Transactions of the Association for Computational Linguistics 1 (2013), 49–62. https://doi.org/10.1162/tacl_a_00020

[10] Yoav Artzi and Luke S. Zettlemoyer. 2015. Situated understanding and learning of natural language. Ph.D. Dissertation.

[11] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. 2017. DeepCoder: Learning to Write Programs. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.

[12] Laura Banerescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for Sembanking. In Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse. Association for Computational Linguistics, Sofia, Bulgaria, 178–186. https://www.aclweb.org/anthology/W13-2322

[13] Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. 2018. Neural Code Comprehension: A Learnable Representation of Code Semantics. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (Montréal, Canada) (NIPS’18). Curran Associates Inc., Red Hook, NY, USA, 3589–3601.

[14] Chris Callison-Burch, Miles Osborne, and Philipp Koehn. 2006. Re-evaluating the Role of Bleu in Machine Translation Research. In 11th Conference of the European Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, Trento, Italy. https://www.aclweb.org/anthology/E06-1032

[15] Donald D. Chamberlin and Raymond F. Boyce. 1974. SEQUEL: A Structured English Query Language. In Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control (Ann Arbor, Michigan) (SIGFIDET ’74). Association for Computing Machinery, New York, NY, USA, 249–264. https://doi.org/10.1145/809296.811515

[16] Shohit Chaurasia and Raymond J. Mooney. 2017. Dialog for Language to Code. In Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers). Asian Federation of Natural Language Processing, Taipei, Taiwan, 175–180. https://www.aclweb.org/anthology/I17-2030

[17] Wanxiang Che, Yanqiu Shao, Ting Liu, and Yu Ding. 2016. SemEval-2016 Task 9: Chinese Semantic Dependency Parsing. In Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016). Association for Computational Linguistics, San Diego, California, 1074–1080. https://doi.org/10.18653/v1/S16-1167

[18] Xinyun Chen, Chang Liu, and Dawn Song. 2018. Tree-to-Tree Neural Networks for Program Translation. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (Montréal, Canada) (NIPS’18). Curran Associates Inc., Red Hook, NY, USA, 2552–2562.

[19] James Clarke, Dan Goldwater, Ming-Wei Chang, and Dan Roth. 2010. Driving Semantic Parsing from the World’s Response. In Proceedings of the Fourteenth Conference on Computational Natural Language Learning, Association for Computational Linguistics, Uppsala, Sweden, 18–27. https://www.aclweb.org/anthology/W10-2903

[20] Pradeep Dasigi, Matt Gardner, Shikhar Murty, Luke Zettlemoyer, and Eduard Hovy. 2019. Iterative Search for Weakly Supervised Semantic Parsing. 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). Association for Computational Linguistics, Minneapolis, Minnesota, 2669–2680. https://doi.org/10.18653/v1/N19-1273

[21] Li Dong and Mirella Lapata. 2016. Language to Logical Form with Neural Attention. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Melbourne, Australia, 2669–2680. https://doi.org/10.18653/v1/P16-1004

[22] Li Dong and Mirella Lapata. 2018. Coarse-to-Fine Decoding for Neural Semantic Parsing. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Melbourne, Australia, 731–742. https://doi.org/10.18653/v1/P18-1068
