DSL-Xpert: LLM-driven Generic DSL Code Generation

Daniel Garcia-Gonzalez[*]
Universidade da Coruña
A Coruña, Spain
d.garcia2@udc.es

Victor Lamas
Universidade da Coruña
A Coruña, Spain
victor.lamas@udc.es

Miguel R. Luaces
Universidade da Coruña
A Coruña, Spain
miguel.luaces@udc.es

ABSTRACT

Nowadays, large language models (LLMs) are an extremely useful and fast tool to complement and help in many jobs and current problems. However, there are cases where a pretty specific vocabulary is used in which these models were not previously trained, leading to less satisfactory results. More specifically, these models are less effective when dealing with less-known or unpublished domain-specific languages (DSLs). Within this field, the automatic generation of code based on such languages, starting from natural language, would speed up the development times of any related project, as well as the understanding of such DSLs. Therefore, this paper presents a tool in which developers can perform what is known as semantic parsing. In other words, the developer can ask a pre-trained LLM to translate a natural language instruction into the vocabulary of the established DSL. Thus, by setting the DSL grammar as context (grammar prompting) and providing usage examples (few-shot learning), the LLM can quickly generate reliable domain-specific code, significantly improving the quality of life of the developers. A video demonstration of the tool is shown in the following link: https://zenodo.org/records/12610506.

CCS CONCEPTS

• **Software and its engineering → Application specific development environments**.

KEYWORDS

Domain-specific languages (DSLs), large language models (LLMs), semantic parsing, grammar prompting, few-shot learning

ACM Reference Format:
Daniel Garcia-Gonzalez, Victor Lamas, and Miguel R. Luaces. 2024. DSL-Xpert: LLM-driven Generic DSL Code Generation. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3652620.3687782

1 INTRODUCTION

Within the software engineering field, the development of domain-specific languages (DSLs) is a critical point in order to express and manage domain-specific concepts and processes clearly, especially for people less familiar with coding. Unlike general-purpose languages, DSLs are designed to address each domain's particular requirements and complexities, offering greater expressiveness, abstraction, and efficiency. In addition, DSLs also promote reusability, maintainability, and scalability, thus optimizing workflows and improving software quality.

However, despite all these benefits, DSLs are not without drawbacks. One significant challenge is the overhead cost associated with the design, implementation, and maintenance of each individual DSL, which typically requires specialized expertise and longer development times. In addition, the lack of robust support for each tool and comprehensive documentation can increase the learning curve for each DSL, discouraging potential users and restraining the widespread diffusion of this kind of solution.

For these reasons, the emergence and far-reaching use of large language models (LLMs) represents a paradigm shift in DSL engineering. These LLMs, such as GPT-3 and its successors, are models already trained using vast amounts of natural language data, which gives them a deep understanding of linguistic structures, patterns, and semantics. In this way, LLMs can facilitate various aspects of DSL development, such as code generation and documentation, thus optimizing their lifecycle and improving developer productivity.

Nonetheless, these LLMs need prior configuration in order to work accurately with each particular DSL. After all, although these have been trained with large amounts of data and, in many cases, some well-known DSLs such as SQL, most of the DSLs are usually not public or previously known. To overcome this issue, this paper presents DSL-Xpert, a tool focused on using GPT-based LLMs for code generation for previously defined DSLs. Along with a previous context configuration of the LLM and its usage parameters, using grammar prompting and few-shot learning, the tool is able to perform semantic parsing towards a specific DSL. In this way, there is no need for an exhaustive knowledge of it.

The paper is organized as follows. Section 2 presents a number of recent related studies, as well as the concepts needed to understand how the tool works. Then, in Section 3, the specific functionalities of the tool, as well as its architecture, are discussed. After that, in Section 4, a representative usage example is shown, with each of the necessary steps to carry it out. Finally, Section 5 reviews some limitations of the tool and Section 6 presents a series of conclusions and future work lines.

2 BACKGROUND

This section presents works related to the automatic generation of DSL-specific code. To this end, four key concepts are introduced and detailed in the following subsections. However, it should be noted that the use of LLMs to deal with DSL-related problems is an extremely recent topic with a long way to go. However, there

[*]All authors have contributed equally and the names are listed in alphabetical order.

MODELS Companion '24, September 22–27, 2024, Linz, Austria
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3687782

are works with remarkably diverse approaches, from the direct use of ChatGPT [5] to the study of the results when using different LLMs, prompts, and types of DSLs [7, 15]. In the case of DSL-Xpert, the tool presented in this paper, it takes advantage of all these new features already provided in previous works, resulting in an efficient, transparent, and easy-to-use tool.

2.1 Semantic parsing

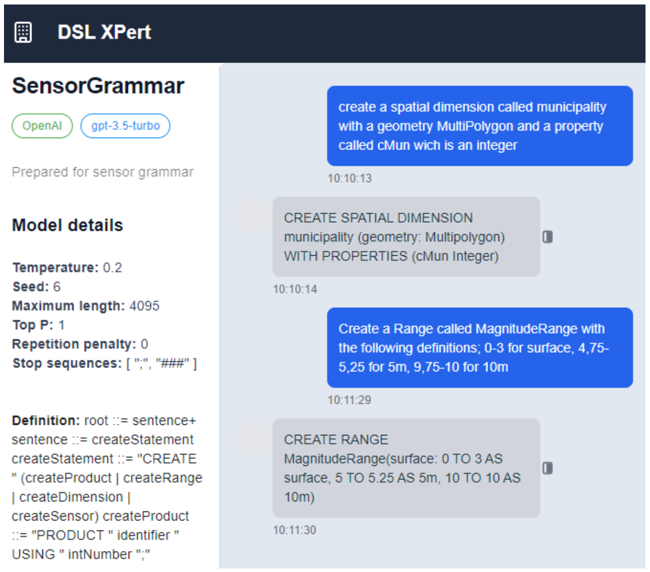
Given the huge proliferation of LLMs in recent years, the semantic parsing field is being extensively studied [8]. More specifically, this process consists of understanding the meaning behind each linguistic expression introduced. Within the scope presented in this paper, it would involve interpreting the user's intentions when writing instructions related to the DSL in question. Thus, for example, if the model is asked to "create a new product", the LLM should be able to translate this concept using the DSL previously specified, without performing the literal action requested in that instruction. Fortunately, with the broad supply of LLMs available and already trained with vast amounts of data and in different scenarios, this is a problem that can be avoided for the topic of this paper. The resources needed to use and create a custom model that could work similarly to these are absurdly large. In this way, few research groups could carry it out. Given that, models such as OpenAI's GPT [2] or Meta's LLaMa [1] could already be used to address this problem with excellent results.

2.2 Grammar prompting

To ensure that the LLM correctly understands what to do and performs a successful semantic parsing, it is essential to previously establish the rules to be followed or, in this case, the specific vocabulary to be used. In many cases, this is not mandatory, since the available models are usually already trained to work correctly for generic purposes. However, there are specific cases, such as DSL-specific code generation, where this step is fundamental. Given that, grammar prompting would consist of previously defining the syntactic and semantic rules that define the structure and behavior of the DSL [14]. One of the most common approaches in the literature to represent these grammars is the Backus-Naur Form (BNF) [10, 11], since it provides a clear and easy-to-understand definition. Nevertheless, other approaches like those based on JSON or XML could also be used with high-grade results [4, 16]. In this way, the LLM can recognize the vocabulary represented in those grammars, resulting in responses that use the patterns therein.

2.3 Few-shot learning

One of the most common dynamics when it comes to quickly establishing contexts to be followed by LLMs is the so-called few-shot learning [12]. In this case, the model is intended to be able to generate new instances from a series of previous examples of use, assimilating and detecting the common patterns in these samples. Therefore, the more examples and the more varied they are, the better the generation capability of the final model [3]. For the case discussed in this paper, each of these examples would consist of the instruction to be translated to the DSL in question, written in natural language, together with the expected response of the LLM, using the grammar of that DSL. Thus, it is worth noting that this approach has many limitations, since it greatly depends on the examples provided. If an attempt was made to infer a significantly different example from those provided before, the model could give inaccurate results. In any case, it is a very fast and easy-to-apply dynamic that is worth using in developments of this type.



SensorGrammar

Prepared for sensor grammar

Model details

Temperature: 0.2
Seed: 6
Maximum length: 4095
Top P: 1
Repetition penalty: 0
Stop sequences: [";", "###"]

Definition: root ::= sentence+ sentence ::= createStatement createStatement ::= "CREATE " (createProduct | createRange | createDimension | createSensor) createProduct ::= "PRODUCT " identifier " USING " intNumber ";"

create a spatial dimension called municipality with a geometry MultiPolygon and a property called cMun wich is an integer
10:10:13

CREATE SPATIAL DIMENSION municipality (geometry: Multipolygon) WITH PROPERTIES (cMun Integer)
10:10:14

Create a Range called MagnitudeRange with the following definitions; 0-3 for surface, 4,75-5,25 for 5m, 9,75-10 for 10m
10:11:29

CREATE RANGE MagnitudeRange(surface: 0 TO 3 AS surface, 5 TO 5.25 AS 5m, 10 TO 10 AS 10m)
10:11:30

Figure 1: Example of a model already created in DSL-Xpert, with its parameters configured and the LLM chat running.

2.4 Model fine-tuning

As discussed previously, the LLMs available to the general public work exceptionally well in most situations. However, this generic purpose means they do not work as well when dealing with pretty unique cases such as the one discussed in this paper. For this reason, one of the options that many developers use is to fine-tune these pre-trained models [6, 13]. In this way, in addition to the large amounts of data already processed by these models, a set of data related to the given subject would be added, in which the LLM would be trained again to incorporate it into its internal knowledge. Within this field, one of the most common approaches is Retrieval-Augmented Generation (RAG) [9]. Here, instead of adapting the selected model to the given context, a series of external data is added to it, which can then be employed to enrich its answers, facilitating and accelerating the process. In any case, the resources required for fine-tuning models are greater than those of the previously mentioned approaches. A large and representative data set would be needed to obtain somewhat satisfactory results, as well as equipment powerful enough to carry out the training in a reasonable time.

3 LLM INTEGRATION

This section details all the functionalities of DSL-Xpert, as well as the general architecture of the tool. The whole development was based on the following fundamental objectives:

- Taking advantage of the new technologies based on artificial intelligence and improving their use for domains in which they are not as excellent.
- Bringing closer and simplifying the use of DSL and LLM approaches for people less familiar with coding.
- Elaborating a flexible, intuitive, and practical tool that works accurately independently of the specified domain.

3.1 Features

Taking into account the objectives highlighted before, the main features of the tool focus on the following two points:

- **Model creation**. DSL-Xpert offers the possibility to create different models for each related problem. In this way, the user can apply different types of DSLs or configurations of the selected LLM, depending on the associated context. Thus, a ready-to-use agent is available, both for the creation of new code and for helping new users understand the specific domain.
- **LLM chat**. Once a model has been created and all its parameters configured, the tool establishes a direct communication channel between the user and the selected LLM. Hence, each of the instructions entered is quickly translated by the model into the DSL in question.

To see how the tool looks with these functionalities, Figure 1 shows the chat layout, as well as the parameters defined for the created model. As can be seen, the instructions entered by the user are almost immediately transferred to the vocabulary of the designated DSL, with all the related advantages discussed in the previous paragraphs.

3.2 Tool architecture

DSL-Xpert was built from the following components:

- **Client.** The tool's architecture comprises a client-side visual component developed using Vue 3 (a Javascript Framework for building user interfaces), Vuetify (a visual component library for Vue), and Vue Router to facilitate navigation between views and components.
- **Server.** The tool exposes a RESTful API for CRUD operations on models and chat functions on the server side by combining Node.js and Express.js. It facilitates CRUD operations by defining database schemas using Mongoose, a MongoDB object modeling tool. The server also creates requests to the OpenAI API when needed (normally when the user sends a request to a model).
- **Database.** DSL-Xpert uses MongoDB, a non-relational database, to persist data that is accessed and modified by server-side operations.
- **Docker.** The client, server, database, and Nginx configurations are all contained within distinct containers thanks to the tool's Dockerization of each component. Similarly, an Nginx container manages the redirection between client and server components. Because the Docker Compose configuration exposes a single route for the final application, this method simplifies deployment and ensures consistency across various environments.

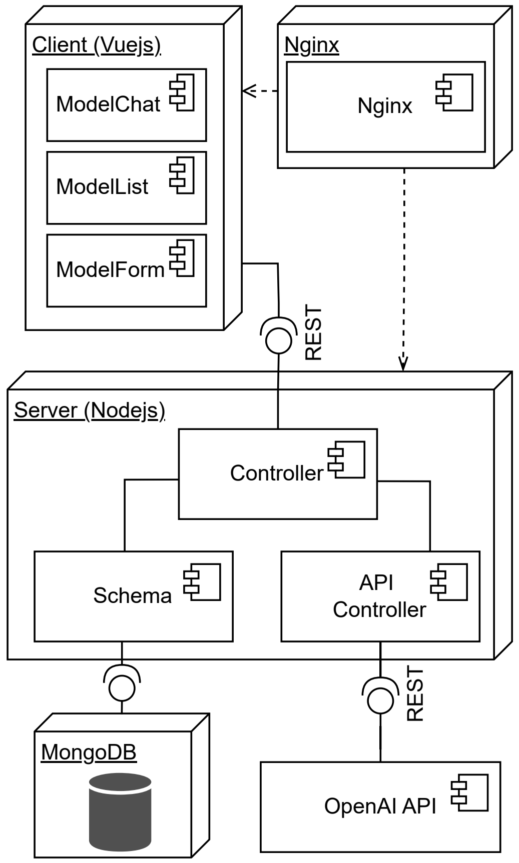


Figure 2: DSL-Xpert architecture.

This architecture is depicted in Figure 2, with the server handling persistence and communicating with the MongoDB database. Likewise, the visual client sends requests to the server via the REST protocol. As can be seen, each Docker container managed by Docker Compose is represented by the blue dotted lines.

4 RUNNING EXAMPLE

To prove the usefulness of the tool in a much more visual way, this section presents a running example, highlighting the reliability of the results provided by the LLM. To set up the initial configuration and be able to reproduce the complete example, the next seven steps must be followed:

1. Download the source code from https://github.com/lbdudc/dsl-xpert.
2. Set the OpenAI API key to your one inside the *.env* file (previous account creation is required).
3. Ensure that Docker is installed before executing the tool by typing `docker-compose up` in the terminal (additional options to run the tool are detailed in the project's README).
4. Go to *localhost:5555* in your browser to access the tool and start creating a new model by clicking on "New model".
5. Set the model name and description. Also, choose as developer "OpenAI", and as model type "gpt-3.5-turbo". The rest of

the parameters can be the default, except for model grammar and usage examples.

(6) Write the following grammar inside the model grammar parameter:

Listing 1: Grammar for a DSL that describes IoT datawarehouses

```
root ::= createStatement+
createStatement ::= "CREATE " (createProduct |
    createRange | createDimension | createSensor)
createProduct ::= "PRODUCT " identifier "USING"
    intNumber ";"
createRange ::= "RANGE " identifier "(" rangeProp (","
    rangeProp)* ");"
rangeProp ::= rangeNumber ( "TO " rangeNumber)? "AS "
    identifier ( "COLOR " hexColor)? | "DEFAULT " "AS "
    identifier ( "COLOR " hexColor)?
rangeNumber ::= identifier | floatNumber | intNumber |
    "INFINITY" | "-INFINITY"
createDimension ::= "SPATIAL DIMENSION " identifier "("
    "geometry" ":" (type | "Geometry") ")"
    createDimProp ";"
createDimProp ::= "WITH PROPERTIES (" dimPropDef (","
    dimPropDef)* ")"
dimPropDef ::= identifier type
createSensor ::= "SENSOR " identifier "(" "interval" ":
    " intNumber "," "datasource" ":" dataSource ","
    "geometry" ":" type ")" createSensorProp
    sensorMeasData (sensorSpatDim)* ";"
createSensorProp ::= "WITH PROPERTIES (" sensorProp (","
    sensorProp)* ")"
sensorProp ::= identifier type
sensorMeasData ::= "WITH MEASUREMENT DATA (" measProp
    ("," measProp)* ")"
measProp ::= identifier type ( " UNITS " identifier)? (
    " ICON " identifier)? (" RANGE " identifier)?
sensorSpatDim ::= "WITH SPATIAL DIMENSIONS " identifier
    "("
identifier ::= text
char ::= [a-z] | [A-Z] | [0-9]
text ::= char | char text
type ::= "Long" | "Boolean" | "Float" | "Integer" | "
    Double" | "Localdate" | "String" | " Datetime" |
    "Linestring" | "Polygon" | "Point"
dataSource ::= "postgres" | "elasticsearch"
hexColor ::= "#" dig dig dig dig dig dig
dig ::= [0-9] | [A-F]
```

(7) Enter the following usage examples at the end of the model creation interface (first the model instruction and then the expected result). For each usage example, we first give the example user instruction, and then expected model answer.

Listing 2: First usage example

```
Create a product called intecmar with an srid of 4326.

CREATE PRODUCT intecmar USING 4326;
```

Listing 3: Second usage example

```
Create a spatial dimension called Municipality with a
    Geometry geometry and a property called cMun, which
    is an Integer.

CREATE SPATIAL DIMENSION Municipality (geometry:
    Geometry) WITH PROPERTIES (cMun Integer);
```

Listing 4: Third usage example

```
Create a range called MagnitudRange with the following
    definitions: 0-3 for surface, 4.75-5.25 for 5m and
    9.75-10.25 for 10m.

CREATE RANGE MagnitudRange (0 TO 3 AS "surface",4.75 TO
    5.25 AS "5m",9.75 TO 10.25 AS "10m");
```

Listing 5: Fourth usage example

```
Create a sensor called StationObservation with a 300
    interval, an elasticsearch datasource and a Point
    geometry. It also has several different properties
    called as follows, with their datatype between
    parenthesis: name (String) and desc (String). It
    also has the following measurement data names,
    along with their datatypes, units, icon and range
    between parenthesis (note that many of them may
    lack the range and the icon information): temp
    (Double, "K", "therm", TempRange), salinity
    (Double, "PSU"), pressure (Double, "dbar"), ph
    (Double, "pH"). It also has one spatial dimension
    called Estuary, which includes the Estuary
    dimension.

CREATE SENSOR StationObservation (interval: 300,
    datasource: elasticsearch, geometry: Point) WITH
    PROPERTIES (name String, desc String) WITH
    MEASUREMENT DATA (temp Double UNITS "K" ICON
    "therm" RANGE TempRange, salinity Double UNITS
    "PSU", pressure Double UNITS "dbar", ph Double
    UNITS "pH") WITH SPATIAL DIMENSIONS Estuary (
    Estuary );
```

(8) After the model has been created with the DSL grammar specified and examples of inputs and expected outputs added, the user can begin entering inputs to ask the model to create some DSL instances through chat. The following examples show the text entered in the model's chat window and the actual outcomes that were achieved with highlighted syntax.

Listing 6: First chat example

```
Create a product called 'MITCGIT' with a srid of 23568

CREATE PRODUCT MITCGIT USING 23568;
```

Listing 7: Second chat example

```
Write a spatial dimension 'Municipality' with a geometry
    of type geometry and a property called 'cMun',
    which is an Integer.

CREATE SPATIAL DIMENSION Municipality (
   geometry: Geometry
) WITH PROPERTIES (
   cMun Integer );
```

Listing 8: Third chat example

```
Formulate a range known as 'Depth' that includes the
    following values: 0-3 (shallow), 4-9 (deep), 10-20
    (very deep).

CREATE RANGE Depth (
   0 TO 3 AS "shallow",
   4 TO 9 AS "deep",
   10 TO 20 AS "very deep" );
```

Listing 9: Fourth chat example

```
Create a sensor called 'Traffic' with an interval of
      300, made in postgres and a point geometry. It has
      a series of properties defined below: desc (String)
      and name (String). As for the measurement data,
      they are presented below, together with their
      typology and units in brackets: intensity (Double,
      'veh/h'), speed (Double, 'km/h'), occupancy
      (Double, '\%'). Finally, this sensor belongs to two
      different spatial dimensions: one called
      'Administration' which includes 'Section',
      'District' and 'Municipality'; and another called
      'Neighbourhood' which includes the dimension called
      'Neighbourhood'.

CREATE SENSOR Traffic (
     interval: 300,
     datasource: postgres,
     geometry: Point
) WITH PROPERTIES (
     desc String,
     name String
) WITH MEASUREMENT DATA (
     intensity Double UNITS "veh/h",
     speed Double UNITS "km/h",
     occupancy Double UNITS "%"
) WITH SPATIAL DIMENSIONS Administration (
     Section, District, Municipality
) WITH SPATIAL DIMENSIONS Neighbourhood (
     Neighbourhood );
```

5 LIMITATIONS AND THREATS TO VALIDITY

Despite its strength, DSL-Xpert has some drawbacks and validity issues. First, the quality and thoroughness of the grammar and the examples supplied determine how effective the tool is. In fact, the resulting code may be inaccurate if these are missing any information or are not entirely representative. Furthermore, the tool's performance might vary notably based on the specificity and complexity of the DSL, and its generalizability across other DSLs is restricted, requiring adjustments for each new project. In addition, due to the reliance on pre-trained LLM models, the outcomes may be impacted by any biases or restrictions present in these models. Hence, the setting of all the related parameters is crucial.

Finally, the tool's output has not been strictly evaluated. That means that more comprehensive, quantitative and qualitative testing and a range of different evaluation scenarios are needed to assess its resilience and reliability in real-world applications.

6 CONCLUSIONS AND FUTURE WORK

This paper presents DSL-Xpert, a tool qualified to generate code for a specific DSL quickly and easily, without the need to be an expert in the domain. Thus, the tool establishes a direct channel with the OpenAI API, creating a chat using a previously selected GPT model. Each of these channels is set up beforehand, independently for each specific project. In this way, the different parameters that shape the chat are defined. In addition to the parameters of any current LLM, the initial context of the chosen LLM is also established, through the specific DSL grammar and some usage examples. Hence, an environment is created in which the LLM is able to translate the instructions received into the vocabulary defined by the project's DSL in a matter of seconds. Consequently, in addition to the evident gain in time, it also helps to make the DSLs' learning curve much smoother for people more inexperienced in the domain.

Currently, work is underway to increase the flexibility of DSL-Xpert, by introducing other LLM types, in addition to those already available from OpenAI. As a result, it would facilitate access to many more users, being able to use models from different brands and even custom ones. Consequently, it could potentially improve the final results by using more specific and pre-trained models in dynamics more related to the problem in question.

ACKNOWLEDGMENTS

CITIC is funded by the Xunta de Galicia through the collaboration agreement between the Department of Culture, Education, Vocational Training and Universities and the Galician universities for the reinforcement of the research centers of the Galician University System (CIGUS); partially funded by MCIN/AEI/10.13039/501100011033 and "NextGenerationEU"/PRTR: [PLAGEMIS: TED2021-129245B-C21]; partially funded by MCIN/AEI/10.13039/501100011033 and EU/ERDF A way of making Europe: [EarthDL: PID2022-141027NB-C21]; partially funded by GAIN/Xunta de Galicia: [GRC: ED431C 2021/53 and ED431G 2023/01]

REFERENCES

[1] [n. d.]. Meta Llama models. https://github.com/meta-llama. Accessed: 06/05/2024.
[2] [n. d.]. OpenAI models. https://platform.openai.com/docs/models. Accessed: 06/05/2024.
[3] Nils Baumann, Juan Sebastian Diaz, Judith Michael, Lukas Netz, Haron Nqiri, Jan Reimer, and Bernhard Rumpe. 2024. Combining Retrieval-Augmented Generation and Few-Shot Learning for Model Synthesis of Uncommon DSLs. In *Modellierung 2024 Satellite Events*. Gesellschaft für Informatik eV, 10–18420.
[4] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2024. Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation. *arXiv preprint arXiv:2403.06988* (2024).
[5] Daniel Busch, Gerrit Nolte, Alexander Bainczyk, and Bernhard Steffen. 2023. ChatGPT in the loop: a natural language extension for domain-specific modeling languages. In *International Conference on Bridging the Gap between AI and Reality*. Springer, 375–390.
[6] Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146* (2018).
[7] Rijul Jain, Wode Ni, and Joshua Sunshine. 2023. Generating Domain-Specific Programs for Diagram Authoring with Large Language Models. In *Companion Proceedings of the 2023 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. 70–71.
[8] Aishwarya Kamath and Rajarshi Das. 2018. A survey on semantic parsing. *arXiv preprint arXiv:1812.00978* (2018).
[9] Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. 2022. A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110* (2022).
[10] Daniel D McCracken and Edwin D Reilly. 2003. Backus-naur form (bnf). In *Encyclopedia of Computer Science*. 129–131.
[11] Katsumi Okuda and Saman Amarasinghe. 2024. AskIt: Unified Programming Interface for Programming with Large Language Models. In *2024 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 41–54.
[12] Archit Parnami and Minwoo Lee. 2022. Learning from few examples: A summary of approaches to few-shot learning. *arXiv preprint arXiv:2203.04291* (2022).
[13] Kushala VM, Harikrishna Warrier, Yogesh Gupta, et al. 2024. Fine Tuning LLM for Enterprise: Practical Guidelines and Recommendations. *arXiv preprint arXiv:2404.10779* (2024).
[14] Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A Saurous, and Yoon Kim. 2024. Grammar prompting for domain-specific language generation with large language models. *Advances in Neural Information Processing Systems* 36 (2024).
[15] Jiaye Wang. 2024. Guiding Large Language Models to Generate Computer-Parsable Content. (2024).
[16] Idrees A Zahid and Shahad Sabbar Joudar. 2024. Enhancing XML-based Compiler Construction with Large Language Models: A Novel Approach. *Mesopotamian Journal of Big Data* 2024 (2024), 23–39.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009