IEA-Plot: Conducting Wafer-Based Data Analytics Through Chat

Matthew Dupree, Min Jian Yang, Yueling (Jenny) Zeng, Li-C. Wang
University of California, Santa Barbara
Santa Barbara, California 93106

***Abstract*—This paper presents key ideas behind IEA-Plot, a software framework designed to conduct test data analytics through chat. We use wafer-based data analytics as an application example to discuss the ideas. IEA-plot interacts with a user through a dialog and produces plots according to user instructions. At the core of IEA-Plot is a knowledge graph connecting a frontend natural language parser to a backend API. This knowledge graph captures our analytics knowledge in the specific context. Usage examples are presented based on test data collected from a recent production line.**

1. Introduction

The emergence of large Language Models (LM) [1][2][3][4][5] has changed our view for implementing a domain-specific AI software tool like IEA (IEA stands for Intelligent Engineering Assistant, see [6][7]). Among them, ChatGPT [5] has demonstrated remarkable performance for engaging in dialog on a wide variety of topics, including answering questions and generating code.

Figure 2 shows an example dialog with ChatGPT regarding how to correlate wafer map pattern to E-test parameter. The first question asks a *how-to* question, followed by a question asking to list the relevant statistical methods. The third question asks for a *list of steps* to perform the correlation. It can be seen from the responses that ChatGPT presents a very good general understanding of the topic.

In contrast to a general dialog like that shown in Figure 2, our use of a LM is more specific. We assume that user inputs are about how to perform a certain task in a test data analytics context. More importantly, we require model responses to be confined by a set of pre-defined actions.

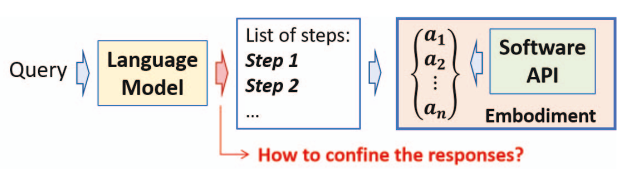

Figure 1. Task grounding problem: How to confine model responses within the scope of an embodiment with admissible actions $\{a_1, a_2, \ldots, a_n\}$?

Figure 1 illustrates the *task grounding* problem. First there is a given *embodiment* for performing the tasks. In our application, the embodiment is based on a software API which defines a set of *admissible actions* $\{a_1, \ldots, a_n\}$. While a model response has to be a list of steps, each step has to be *realizable* by a subset of the admissible actions.

1.1. The task grounding problem

Task grounding means that one desires to use a large LM to *act* in a specific environment [8][9]. Given a LM learned with rich world knowledge, the goal is to ground high-level instructions expressed in natural language to a defined set of actions admissible for a given embodiment. Simply put, we desire to constrain the responses from the LM to be within the capability of the given embodiment.

Consider the response of question 3 in Figure 2. Step 4 suggests Pearson's correlation and Spearman's rank correlation. In our specific context, "*correlation*" can have a different meaning. For example, the "*correlate*" in "correlate a wafer map pattern to E-test parameter" means *to find an E-test parameter whose values can be used to indicate the possible occurrence of the wafer map pattern during wafer sort* (i.e. this is called "failure pattern feedback" in [10]). By grounding a LM with this domain knowledge, we desire the LM to interpret the term "*correlation*" as how we would interpret it (not how a common person would).

Steps like those listed with question 3 also need to be specific enough to enable automatic mapping to some API calls for their realization. By grounding a LM with a set of admissible actions, we ensure each step in LM's responses to be always realizable.

1.2. Usual approaches

One popular way to constrain LM's responses with a specific response structure is through *prompt* engineering [2][4]. Prompt engineering provides input-output examples with a task specifier (the "prompt") for the model to emulate the desired response structure. However, as pointed out in [9], prompt engineering alone is not sufficient to fully constrain a LM to a set of admissible actions.

In addition to prompt engineering, there are two usual approaches to constrain LM's responses. One is to use *reinforcement learning* to align the responses to the desired outputs [8]. Another way is by treating it as a *constrained semantic parsing* problem [11] and implementing a way to adjust/select responses such that only admissible actions are allowed [9]. It can be implemented as a postprocessing component separated from the LM, e.g. to rank/score responses from LM and only admit those acceptable ones.

1.3. Feasibility of the usual approaches

Prompt engineering would be effective if we can be sure that the LM has the desired knowledge or skills in the model,

2378-2250/23/$31.00 ©2023 IEEE
DOI 10.1109/ITC51656.2023.00028

Regular Paper

Q1: Please explain how I can **correlate** a **wafer map pattern** to the **E-test parameter**.
A: In semiconductor manufacturing, a **wafer map** is a graphical representation of a silicon wafer that shows the location and characteristics of individual dies (chips) on the wafer. **E-test** is a testing method used to identify defects in individual dies on a wafer.

To correlate a wafer map pattern to the E-test parameter, you need to first understand the E-test parameter being measured. E-test parameters can include things like leakage current, threshold voltage, and other electrical properties.

Once you have identified the E-test parameter being measured, you can then look at the wafer map to see if there is any correlation between the location of the dies on the wafer and the E-test parameter. For example, you may notice that dies in a certain area of the wafer consistently have higher leakage current values than other areas of the wafer.

To further investigate the correlation, you can use statistical methods to analyze the data and determine if there is a significant correlation between the wafer map pattern and the E-test parameter. This analysis can help identify any patterns or trends that may be indicative of manufacturing issues or other problems that need to be addressed.

Q2: Can you point to me what **statistical methods** are useful in this context?
A: Certainly! There are several statistical methods that can be used to analyze the correlation between a wafer map pattern and E-test parameters. Here are a few examples:
1. Correlation analysis: …
2. Regression analysis:
3. ANOVA: ANOVA (analysis of variance) …
4. Cluster analysis: …
5. Principal component analysis (PCA): …

Q3: Can you **list the steps** to guide me how to perform such correlation analysis?
A: Certainly! Here is a general guide to performing a correlation analysis between a wafer map pattern and E-test parameters:
1. Define the research question: …
2. Collect and prepare the data: …
3. Visualize the data: …
4. Compute the correlation: Calculate the correlation coefficient between the wafer map pattern and E-test parameter. You can use Pearson's correlation coefficient, Spearman's rank correlation coefficient, or other measures of correlation depending on the nature of the data.
5. Test for significance: …
6. Interpret the results: …
7. Consider further analysis: Depending on the results of the correlation analysis, you may want to perform additional analyses such as regression analysis, cluster analysis, or PCA to gain a deeper understanding of the relationship between the wafer map pattern and E-test parameter.

Figure 2. Dialog with ChatGPT (03/26/2023) regarding how to correlate wafer map patterns to E-test parameters

and we just need to be more expressive in our queries so that the model knows to respond accordingly. However, verifying that a given LM indeed has the knowledge and skills we desire, by itself, can be a challenging research problem.

For example, we might desire the correlation task to start with our Minions analysis method [12]. We are not sure to what extent a given LM "understands" the proposed method (even though the LM may have "read" the paper).

As proven in [4] and also in [8] for task grounding, reinforcement learning can be an effective approach to align a LM to respond in a specific domain. However, reinforcement learning requires training data. In our application, if we have a way to generate a large dataset of acceptable user queries to the IEA, then we might consider reinforcement learning. To get to that point we need to first define what user queries are acceptable and build a query generator.

Our prior work [13] follows the *constrained semantic parsing* approach [11]. However, comparing to the task grounding problem in Figure 1, the formulation in [13] is more restricted. In [13], the grounding is achieved through a *grammatical model* that limits the responses to be a set of *canonical utterances*. The approach enables implementation of a constrained parser using prompt engineering with GPT-3. However, the parser does not consider scenarios where one query might be contextually related to the next query. In other words, each query is treated independently.

1.4. Our approach

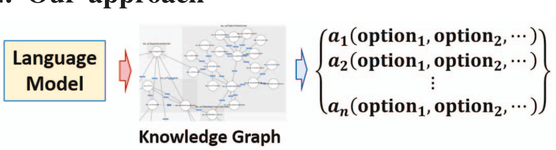

Figure 3. Using a knowledge graph to connect LM and admissible actions

In this work we present an innovative approach for the task grounding problem in our IEA application. Figure 3 depicts the main idea. Our approach uses a *knowledge graph* (KG) [14] to connect a LM and those admissible actions. For each action a_i, it has a set of acceptable options. Hence, for each user query, the responses are confined within the set of actions and their options. The KG serves three purposes:

- From the perspective of the LM, the KG is used to constrain the LM's responses. Each response is represented as a subgraph of the KG.
- From the perspective of backend API, the KG captures our *analytic knowledge* on how to utilize API function calls to accomplish an analytic task.
- From the perspective of IEA implementation, the KG is a model to enable data management for keeping track of the current tool's state during its execution.

If we think that *domain-specific machine learning* (DSML) [15][16] is about "domain knowledge + ML", in our approach the "domain knowledge" is explicitly expressed in the KG, and the ML is involved in the LM and some admissible actions supported in the backend API

For the rest of the paper, section 2 discusses main considerations behind the KG development. Section 3 explains key ideas, considering the three perspectives mentioned above. Section 4 focuses on the approach for pattern analytics, including the task for correlating wafer patterns to E-tests. Section 5 shows the IEA-Plot's results based on a dialog example. Section 6 discusses our constrained semantic parsing approach. Section 7 concludes.

2. Use of Knowledge Graph

Our decision to utilize a KG to connect LM and backend API was partly inspired by a recent trend in natural language research where LM and KG are combined to improve natural language inference (NLI) and question-answer (QA), e.g. see [17][18][19][20][21]. A KG, such as ConceptNet [22], provides structural knowledge that can be used to *ground* the reasoning process through a LM [19]. A KG represents knowledge in a *symbolic space* while a LM represents knowledge in a *vector semantic space*. The major challenge in their work is how to effectively fuse the two representations in a unified manner [20][21].

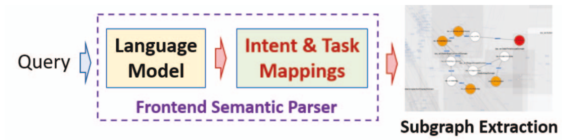

Figure 4. A user query corresponds to a subgraph in the KG

In our IEA design, we mean to use KG in a different way though. As illustrated in Figure 4, the KG provides a target output space for the frontend semantic parser. The parser's job is to map a query to a subgraph of KG. This mapping is based on two aspects of the query: its intent and the implied steps required for the task.

The use of KG was motivated by another objective. We desire to use KG as a central place to store *domain knowledge* and as discussed in Section 1.4, this domain knowledge has three perspectives: (1) the LM perspective (knowledge about the steps involved in an analytic process) (2) the API perspective (knowledge about the analytic tools and their use), (3) the IEA tool perspective (knowledge about the IEA implementation itself). This domain knowledge is expressively represented in the KG and sharing of the knowledge can be achieved by sharing the KG. Ideally, we also want to design the KG such that future scaling of the IEA tool's capability (e.g. adding a new function or option) can be done by adding nodes and edges to the KG.

3. Development of Knowledge Graph

As pointed out in [14], the term "knowledge graph" can have various meanings, and the announcement of the Google KG [23] separates its modern views from the historical views. KG is a rich field. Terms (e.g. the term "ontology") used in the field can sometimes be confusing [14].

In our view, the field includes two distinct uses of KG. One is for representing and organizing the knowledge from vast amounts of data such as data available from the Internet. RDF [24] (for data representation), and RDF Schema (RDFS) and OWL [25] (the ontology languages) are standards for this purpose. The other is for representing and organizing people's knowledge. ConceptNet [22] is a popular example. ConceptNet does not use a complicated ontology (like OWL). It includes 35 well-defined relations to connect common sense "concepts". ConceptNet is suitable for representing the graph nodes with *distributional vector embeddings*, making it suitable for use with a LM to perform joint reasoning [19][20][21].

Given our multiple purposes to use KG from the three perspectives, we need a hybrid model somewhat between RDF/OWL and ConceptNet. On one hand, we need a KG capable of modeling the data in our tool. On the other hand, we need the KG simple enough to enable LM+KG joint reasoning in the future.

3.1. The KG design

To avoid confusion, Figure 5 clarifies the formalism of our KG design and the terminology in use. A graph is a collection of triples $(source,\ predicate,\ target)$. The KG comprises two separate graphs: the *domain graph* and the *data graph*. When we refer to the term *ontology*, we mean the domain graph. The domain graph provides our interpretation to process the data graph, and this processing includes the three perspectives mentioned before.

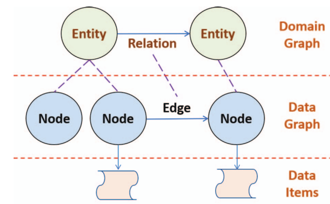

Figure 5. Hierarchy in our knowledge graph design

In the *domain graph*, sources and targets are called *entities*. Predicates are called *relations*, which are always directional. In the *data graph*, sources and targets are *nodes* and predicates are *edges*. A node is an instance of an entity and an edge is an instance of a relation. These usages of the terms are consistent with those described in [14].

A node can have a list of data items. There are two types of data items stored with a node. The first is to store data objects relevant to the execution of the tool. The second is to store example phrases/sentences that represent the semantic meaning of the node.

3.2. Our Ontology

In the field of KG, an *ontology* is a concrete, formal representation of what terms means in the given domain [14]. Figure 6 depicts our ontology as axioms in the domain graph. A triple $(Entity_1, Relation_a, Entity_2)$ means that a node of $Entity_1$ and a node of $Entity_2$ *can* have an edge of $Relation_a$. If a triple $(Entity_1, Relation_b, Entity_2)$ is not present in this graph, then it means the two nodes cannot have an edge of $Relation_b$. Hence, the graph provide the axioms that triples in the data graph have to satisfy.

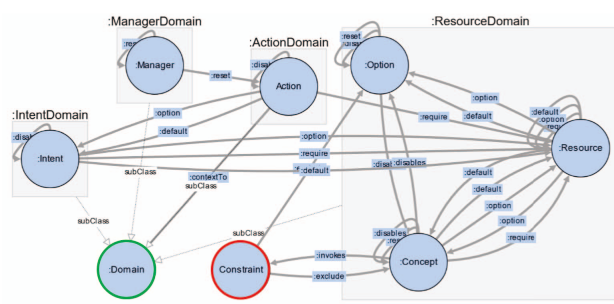

Figure 6. Axioms in the domain graph, representing the ontology we use

The semantics of the domain graph can be explained as the following. We divide nodes into four domains: **Manager**, **Action**, **Intent**, and **Resource**. These four entities are all subclasses of the parent entity **Domain**. Inside the resource domain, there are two types of nodes: **Concept** and **Option**. In addition, there is a special type of node called **Constraint** which is used to capture constraints between concept-concept and concept-option pairs, whenever needed. In total, our ontology defines 8 entities (types of nodes).

The ontology further defines eight types of relations. Below we use the term "activate" to mean that during subgraph extraction, a node is included in the current subgraph.

1) (source, **require**, target): When the source is activated, the target is required to be activated.
2) (source, **default**, target): When the source is activated and no option node is activated by the user query, the target is the default and is activated.
3) (source, **option**, target): The target can be activated as an option when the source is activated.
4) (source, **reset**, target): When the source is activated by the user query, all previously-activated options from the target are reset.
5) (source, **disable**, target): When the source is activated, the target is deactivated. This can be used to model mutually-exclusive activations among nodes.

The sixth relation is called **contextto**. Its usage is specific to the form $(\mathbf{Action}, \mathbf{contextto}, \mathbf{Domain})$. This is used to model *domain switching* in a dialog. For example, when the analytic context switches from analyzing wafer sort data to correlating between wafer sort data and E-test data, it involves a domain switching.

The last two relations are **invoke** and **exclude**, specific for the use to model node-node constraints. (**Concept**, **invoke**, **Constraint**) means activation of the concept node will invoke the constraint. (**Constraint**, **exclude**, node) means the constraint excludes the activation of the node which can only be a node of either **Concept** or **Option**.

3.3. Key ideas for constructing the data graph

Figure 7 uses a conceptual example to illustrate the key ideas for constructing the data graph[1].

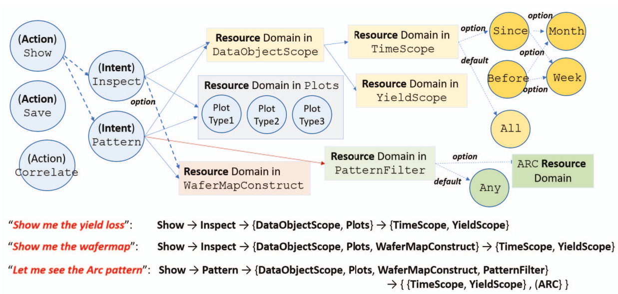

Figure 7. A conceptual example to illustrate our graph construction

Each box in Figure 7 represents a resource domain that contains a subgraph. There are two types of edges shown in the figure. A solid edge is a **require** edge. A dash edge is not a require edge. There are three example queries where for each, the activated subgraph is illustrated.

Consider the first query "*Show me the yield loss*". This query activates the *Show* action node. It is determined (intent determination will be discussed later) that the intent is to *inspect* the data from the "yield loss" perspective. This inspection requires performing two tasks: (1) Select the data scope to inspect. This is represented by the **require** edge pointing to the `DataObjectScope` resource domain. (2) Select a plot type for the display. This is represented by the **require** edge pointing to the `Plots` resource domain.

The data selection task further requires resources from two domains: `TimeScope` and `YieldScope`, where the first allows selecting a time interval based on month or week, while the second allows selecting data based on a yield threshold (not shown in the figure). Because the query does not specify a time scope, the default is `All` which is pointed by the `TimeScope` through a **default** edge.

Similarly, `Plots` domain contains a list of plot types and may include a default (e.g. Type1). Defaults and options are modeled in the subgraph of the `Plots` domain.

The second query requires an additional resource, `WaferMapConstruct`, used to determine how the wafer maps are constructed (e.g. a pass/fail wafermap, a stacked wafermap, a wafermap after some filter, etc.). The second query also requires all resources required by the first query. As a result, all data and options collected from the first query is inherited by the second query as its starting point.

The third query asks to switch *context* from data inspection to do some wafermap pattern analysis. *Context switching* in our design means to switch from one intent to another. The third query requires one additional resource `PatternFilter` (this resource will be discussed in Section 4). Because the query asks to see specifically the "Arc" pattern, this triggers the option to request the `ARC` resource.

3.3.1. The API perspective. From the API perspective, **Action** nodes correspond to the steps in the "main" program. Each resource domain corresponds to a portion of the API functionality. For example, the API support various ways to select the data. How to call those functions with what options are organized in the knowledge subgraph within the `DataObjectScope` domain. In our KG, each node in a resource domain (**Concept** or **Option**) corresponds to an available function in the API. Their dependency structure is modeled in the knowledge subgraph in the domain.

3.3.2. The KG manager perspective. The KG manager is responsible for managing the extracted subgraphs from one query to the next. An extracted subgraph corresponds to a *configuration* telling how to call the backend API.

A configuration contains a list of admissible actions with their options as shown in Figure 3. An action can be thought of as a function call. From the API perspective, function calls are organized in hierarchy (e.g. calling a function a_i may involve calling other functions $a_{i_1}, a_{i_2}, \ldots$). This hierarchy is reflected in the subgraph. In other words, our KG contains the knowledge of the API organization.

From the KG manager's perspective, the three relations, **require**, **default**, **option**, are essential for modeling the functional dependency structure. The three relations, **reset**, **disable**, **contextto**, are used for managing the change of subgraph from one query to the next. Their semantic meanings are illustrated through examples in Figure 8.

1. The data graph is evolving. Our current data graph contains 243 nodes with 1147 edges, which can be accessed from our IEA project page: https://iea.ece.ucsb.edu/iea/project

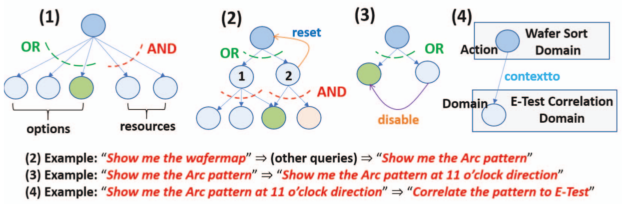

Figure 8. Semantic meanings of the six relations for subgraph management

Example **(1)** in Figure 8 illustrates that a node in an extracted subgraph in general can have two parts of child nodes: the OR part and the AND part. Edges in the OR parts are of **option** type where one of them is **default**. Edges in the AND part are of **require** type. For example, to execute the function represented by the parent, it requires certain specific resources and can have the various options. In this case, we can use **default/option** to model the options and use **require** to model the resources.

Examples **(2)**-**(4)** then illustrate how the KG manager handles subgraph change from one query to the next.

In example **(2)**, the first query asks to see "wafermap". This corresponds to the intent of "wafermap inspection". Suppose node "1" models this intent. After that, there are other queries within the scope of this intent. Then the last query asks to see "Arc pattern", triggering a new intent "pattern analytics". Suppose node "2" models this intent.

Activation of node "2" triggers a *switch of intent*. This is modeled through a **reset** relation from node "2" to its parent. The parent previously maintains the current configuration resulting from queries before the intent switching. The activation of node "2" therefore **reset**s the configuration of its parent, telling it to compute a new one from scratch.

Consider the green node in example **(2)**. It is a shared resource between the two intents. Within the scope of the first intent, those queries may have set the available options under the green node (not shown). Without a reset, a subsequent query would inherit those options. The reset notifies the manager to restores everything back to its default. For the KG manager, options choices are handled cumulatively from one query to the next, until it encounters a **reset**.

Example **(3)** shows a situation where a previously-selected option is replaced with a newly-selected option. The first query asks to see "Arc pattern" but does not specify a direction. Hence, the default is to include *all* directions. This default is the green node in example **(3)**. The second query then provides a specific direction "at 11 o'clock". This option replaces the previous option, and is modeled as a **disable** relation. In general, the **disable** relation can be used to model a set of mutually-exclusive options.

Example **(4)** shows an example of *domain switching*. The first query is in one domain, operating on one dataset, the wafer sort dataset. The second query involves E-test data. In our KG, we consider the two queries belonging to two separate domains. The second query triggers a switch from one-dataset analytics to cross-dataset analytics.

Domain switching is modeled through a **contextto** relation. For example, the **Action** node in the wafer sort domain is labeled as the `Show` action. When the correlation action is recognized, it invokes switching to the E-test correlation domain that contains the action node `Correlate`.

The **contextto** relation tells the manager to (1) bring in the E-test dataset and (2) transfer the *current analytic result* (e.g. pattern groups) from the wafer sort domain to the correlation domain. As discussed in [10], our interpretation of E-test correlation is based on a given set of pattern groups.

3.3.3. The query perspective. Figure 9 shows a sketch of the subgraph for the first query in example **(4)**. As discussed in Section 3.1, example phrases can be attached to a node. Figure 9 shows four such nodes.

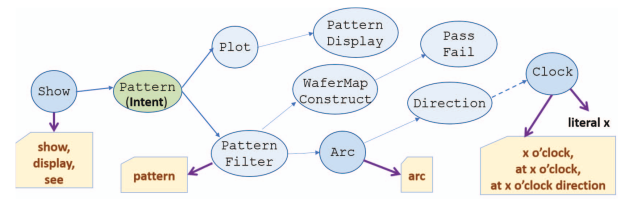

Figure 9. Subgraph for "*Show me the Arc pattern at 11 o'clock direction*"

Given the query, the parser's job is to determine that these four nodes should be activated. Then, the KG manager can extend from the activated nodes to obtain a subgraph (by following the **require** edges).

In a simple way, we might think that the four nodes can be activated by matching texts in the query to the phrases attached with those nodes. For example, *Show* in the query matches "show" in the `Show` node. The text *at 11 o'clock* matches the "at x o'clock" in the `Clock` node. While text matching can be used, it can substantially limit the scope of the acceptable queries to our tool.

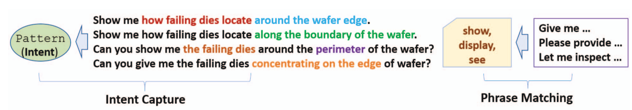

Figure 10. Intent Capture and Phrase Matching for parsing a query

Figure 10 shows that the parser's job includes two aspects: *intent capture* and *phrase matching*. Consider again the first query in example **(4)** in Figure 9. For intent capture, the parser needs to know to activate the `Pattern` node. This might be doable by matching the word *pattern* to the text "pattern" in the `PatternFilter` node. However, consider the four queries shown in Figure 10. Those queries imply to see failing patterns as well, but none of them mentions the word "pattern" or "arc".

Furthermore, the four queries imply looking for a pattern along the wafer edge. If our tool is smart enough, it should know that this includes the `Arc` pattern as we have defined it in our KG. This means that we need to match "arc" with phrases like *wafer edge*, *boundary of the wafer*, *perimeter of the wafer*, etc. We call it the *phrase matching* problem.

Intent capture and phrase matching are the two problems considerd in this work, where we leverage the power of LMs [2][5][26] for tackling the problems. A LM has the ability to "understand" the semantic meaning of a query, to the extent that it helps decide if two queries share the same intent, and if two phrases share the same meaning in view of our KG. In Section 6, we will discuss such use of LMs in detail.

3.3.4. The constraint graph. In our KG, a separate *constraint graph* is maintained using **Constraint** nodes and **invoke/exclude** relations. Figure 11 illustrates their use.

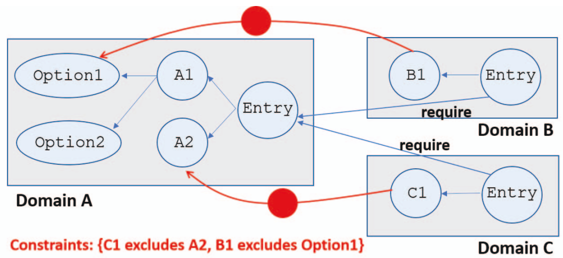

Figure 11. A constraint graph is separately maintained in our KG

In the KG (see footnote 1 in Section 3.3), each resource domain comprises three types of nodes. An `Entry` node is an instance of the **Resource** entity. Inside a domain, an **Option** node is a node with an incoming **option** edge and without any outgoing **option** edge. Other nodes in a domain are instances of the **Concept** entity.

Suppose resource A is shared by both B and C. This is modeled by the two **require** edges from the `Entry` node of B and from the `Entry` node of C, both to the entry node of A. Domain A contains two concepts, `A1` and `A2`, where A1 can have two options, `Option1` and `Option2`. Suppose in our backend API, `B1` can be used with `A1`, but when that happens, cannot have `Option1`. Also, `C1` cannot be with `A2` (if we think `C1` and `A2` as two functions, they cannot be called together due to our API design).

To model such constraints, we create constraint nodes with constraint edges as shown in the figure. The red circles are instances of the **Constraint** entity. Each red edge comprises an **invoke** and an **exclude** relation.

For example, domain A can be resources for making plots. Domain B is for inspecting patterns on wafermaps. Domain C is for inspecting wafermaps in general. `A1` and `A2` are two plot types. While `A1` can be used to display wafermaps, `Option1` is not valid when the wafermaps are given as pattern groups. Also `A2` is a plot that can only be used when wafermaps have already had a pattern group label assigned. To model this knowledge about the API usage structure, we put in the two constraints.

3.3.5. Scalability. An important consideration in our KG design is scalability. Figure 12 shows that extending the tool's functionality can be achieved by adding nodes (and edges) to the KG. This extension can be considered at three levels: **Option**, **Concept**, and **Resource**.

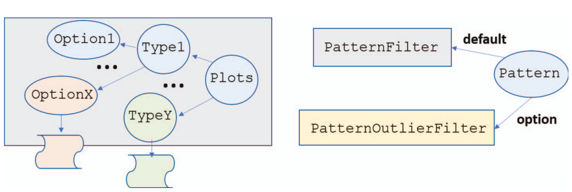

Figure 12. Extending tool functionality by adding nodes in KG

For example, suppose for a particular type of plot we desire to add a new option. With our IEA tool design, we can simply add the option to our API. Then, we add a node `OptionX` to represent this option, and we add a list of example phrases for activation of this option node.

Similarly, if we desire to add a new plot type, we can add a new **Concept** node, say `TypeY` (and its available options as **Option** nodes) in the `Plots` domain, with a list of example phrases for activation of the concept node.

In our current design, the `Pattern` intent node requires resources from the `PatternFilter` domain. For pattern analytics, our current implementation looks for *systematic patterns*. Suppose we desire to include a new way that checks for "outlier" patterns, i.e. a pattern that is both significant and unique. We can add this functionality into the API. On the KG, we can create a new resource domain, say `PatternOutlierFilter`. Then, we can make the `PatternFilter` domain as default and `PatternOutlierFilter` as an option.

As seen in the above three examples, extending the functionality of our tool can be done (mostly) by working within a focused scope of the KG. This locality feature provides a major benefit which facilitates scaling of our tool functionality over time.

4. Pattern Analytics

For wafer pattern analytics, our backend API is built upon the techniques reported previously [6][10][12]. Its core is the *Minions* analysis approach [12]. This section highlights the differences from those previously reported.

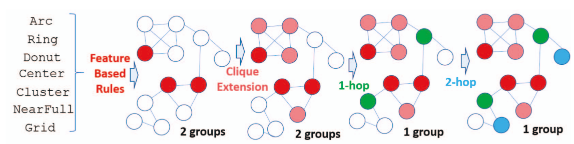

Figure 13. Identifying a pattern group satisfying a pattern `Concept`

As shown in Figure 13, our current implementation includes seven pattern concepts: `Arc`, `Ring`, `Donut`, `Center`, `Cluster`, `NearFull`, and `Grid`. Each pattern concept is modeled as a resource domain in our KG.

Our API supports various preprocessing steps to obtain a wafermap, e.g. salient map, masking, resizing, etc. Those options are modeled in the `WaferMapConstruct` resource domain in our KG. On a wafermap, the value of a die can also be determined in various ways. For example, it can be based on stacking wafers from the same lot, or based on a particular test bin. Those options are modeled in the `WaferMapsRepresentation` resource domain.

Given a set of wafermaps, each is checked to see if it satisfies a pattern concept. This check is based on hard-coded feature-based rules. For the features in use, we experimented and selected features reported from prior works (e.g. [27]) and implemented some of our own [10]. Suppose we want to check if a wafermap contains an `Arc` pattern, in our current approach we first apply the rule-based script for the `Arc` concept. Our rules are designed to be conservative so that if the script considers a wafermap as an `Arc`, the pattern would be obvious from our visual point of view.

After running the script, we obtain a set of `Arc` wafermaps. In our backend, they correspond to some nodes in the *Minions graph*. A Minions graph (see [6][10]) is a graph where nodes are wafermaps, and an edge between two nodes means the two wafermaps are *similar*. This similarity is determined by our Minions analysis approach [12].

Figure 13 provides an example to show that after running a script, three nodes are selected in the Minions graph. At this point, the three nodes are separated in two groups.

Based on the Minions graph, we can extend the set of selected nodes in different ways. First, we extend the selection by including all nodes that belong to a *clique* based on an already-selected node. The result is shown in the second graph in Figure 13 (after the *clique extension*). The selected nodes still form two groups. The first group is a clique of size 4 and the second is a clique of size 3.

The next extension is based on including all nodes that have a direct connection with the current selected nodes. We call this operation the 1-*hop extension*. Those newly-included nodes are marked as green. Then, repeating the idea of 1-hop extension, we can have 2-*hop extension* which includes two additional nodes marked as blue.

4.1. Pattern group

In IEA-Plot, 1-hop extension is the current default to obtain a pattern group. For example, Figure 14 shows the IEA-Plot result based on the query to see the `Center` pattern. The display comprises three levels: after the rules, after the clique extension, and after the 1-hop extension.

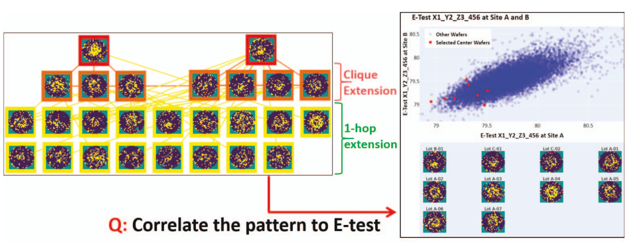

Figure 14. Results based on the `Center` pattern concept

When the next query "Correlate" is given, the pattern group is transferred from the wafer sort domain to the E-test correlation domain. This is the domain switching scenario as discussed with example **(4)** in Figure 8 before. IEA-plot generates correlation plots like the one shown in Figure 14. The plot shows 10 `Center` wafers from 3 lots. The two axes are two sites from the same E-test parameter where the two sites are closest to the pattern. Each dot shows the E-test values of the two sites. Red dots are those 10 wafers.

E-test correlation can result in multiple plots reported. A bias score is assigned to each plot and plots are ranked accordingly. For generating a correlation plot, IEA-plot searches for the subset of wafers from the pattern group, one lot at a time, which give the largest bias score. For detail of this E-test correlation method, please see [10].

Figure 14 shows a scenario where the user already knows what pattern to request. Initially, the user might want to see what patterns are available in the data and what options might be available for a particular pattern.

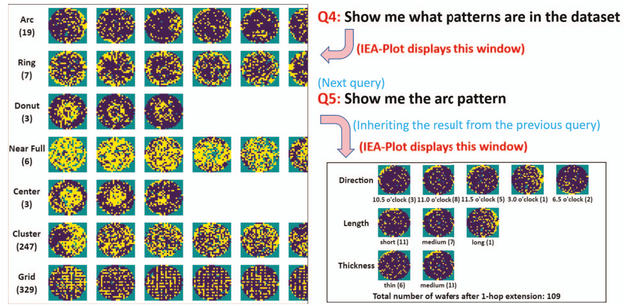

Figure 15. IEA-Plot outputs based on the two consecutive queries

In Figure 15, **Q4** gives a list of available patterns. For the `Arc` pattern, **Q5** gives a list of available options. These results are generated based on the rule-based scripts with no extension. Result of **Q5** shows that the `Arc` pattern can point to five different directions, have three length types, and two thickness types. The number of wafermaps satisfying each option is shown. From here, a user can select options to define a pattern group. If no option from a category is selected in a query, the default is "all".

5. Chat-Based Analytics

Figure 16 shows a dialog example. IEA-Plot's output screenshots for those queries are shown in Figure 17 (except for **Q4** and **Q5** which are already shown in Figure 15 above). The dialog includes an intent switching at **Q4** and a domain switching at **Q12**. These two types of context switching have been discussed in detail above.

Note:	
	Q1: Show me the wafermaps
(Narrowing data scope)	Q2: Show me those only with yield loss greater than #%
(Selecting a new plot type)	Q3: Let me see them based on lot-to-lot arrangement
(Switching Intent; See Figure 15)	Q4: What patterns are in this dataset?
(See Figure 15)	Q5: Show me the options of the Arc pattern
(Pick a pattern group)	Q6: Let's focus on the pattern at 11 o'clock direction
(Selecting a new plot type)	Q7: Let me see which lots they are in
(Narrowing data scope)	Q8: Please zoom-in to January
(Back to previous data scope)	Q9: Go back
(Pick a new pattern group)	Q10: Show me the Donut pattern only
(Back to previous data scope)	Q11: Go back
(Switching Domain)	Q12: Please correlate the pattern to E-test

Figure 16. A dialog example and IEA-Plot's outputs shown in Figure 17

Q2 and **Q8** are two examples where the data scope is narrowed. In IEA-Plot, the data scope is inherited into the next query by default unless (1) the next query selects a new scope, (2) the next query is an intent switching or a domain switching. To restore to the previous data scope, a special query "Go Back" is used. For example, **Q9** resets the data scope back to **Q7** which has the same data scope of **Q6**.

Q10 asks to see a different pattern from **Q6**. This creates a new data scope, i.e. the set of wafers having the `Donut` pattern. **Q11** then resets the data scope back to the previous scope which is the scope of **Q6**. As a result, the `Arc` pattern group is transferred to the E-test domain when processing **Q12**, generating the correlation plot shown with **Q12**.

6. Frontend Semantic Parser

Our parser implementation is fundamentally different from that reported in [13]. With the availability of ChatGPT [28], we leverage its power by taking a *generative approach*. Figure 18 illustrates the approach.

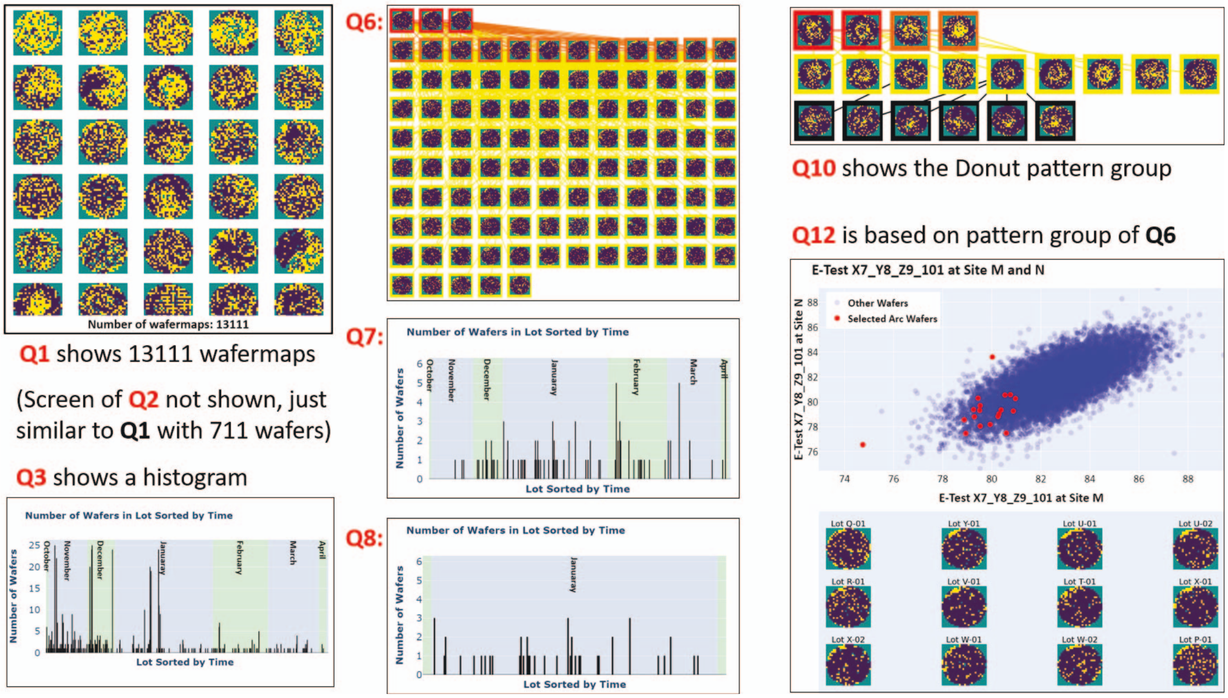

Figure 17. IEA-Plot output screenshots for queries listed in Figure 16, screenshots for **Q4** and **Q5** displayed in Figure 15 previously

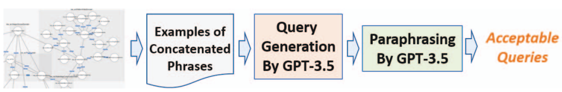

Figure 18. Generating acceptable queries using a GPT model [28]

By traversing the KG, we can extract a list of acceptable configurations (Section 3.3.2). For a configuration, we can generate a list of concatenated phrases (i.e. a sequence of phrases separated by comma) by enumerating combinations of phrases associated with those activated nodes (see Figure 19 for an example). For each concatenated phrase, we ask GPT [28] to make it a full English sentence. Then, we ask GPT to repeatedly paraphrase the sentence to obtain semantically-equivalent sentences. At the end, for each concatenated phrase we obtain a list of acceptable queries.

Concatenated Phrase: { Show, Wafermap display, For all time, In stage2.csv file, Group by lot, Sort by yield loss, With top 100 wafers }

Sentence Completion: Show wafer map display for all time in stage2.csv, grouped by lot, sorted by yield loss, limited to top 100 wafers.

Paraphrasing: Organize wafer map data for all time in stage2.csv, group by lot, sort by yield loss, show top 100 wafers.

Generate a wafermap display for all time periods in the stage2.csv file, grouping the data by lot and sorting by yield loss, with a focus on the top 100 wafers.

Show a wafermap display that groups all time data from stage2.csv file by lot, sorts it by yield loss, and showcases only the top 100 wafers.

Figure 19. An example to obtain a list of acceptable queries

6.1. Intent capture

Parsing of a query is divided into two stages: intent capture and phrase matching (Section 3.3.3). Intent capture determines if the given query requires a *switch of context*. Our KG supports two types of context switching: intent switching and domain switching (e.g. Figure 16). In addition, intent capture determines the current scope of the intent, i.e. the set of allowable nodes in the KG. Under an intent, those nodes are considered for phrase matching.

For intent capture, we use a sentence-BERT (SBERT) model [29]. Figure 20 illustrates our approach. First, we sample a set of acceptable queries where each option is covered at least once. The sampling is done in two phases. First, we sample acceptable configurations. In our experiment, we started with over 250K configurations. Then, we sample a subset of them for generating acceptable queries.

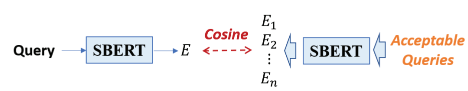

Figure 20. Intent capture by comparing pairwise SBERT embeddings

Given a query, we use SBERT to obtain an embedding, a 384-dimension vector. Given a list of n acceptable queries, we obtain a table of n embeddings ($E_1, \ldots, E_n$ in Figure 20). When a query is entered to IEA-Plot, SBERT generates an embedding E for the query. Then, this E is compared with $E_1, \ldots, E_n$ using cosine similarity. The most similar E_i is used to indicate the intent of the query.

While the approach is simple, it is important to note that its performance can largely depend on the set of acceptable queries stored in the table, i.e. the *coverage* of the set. This coverage depends on the paraphrasing power of the GPT model. IEA-Plot leverages such power provided by the model to simplify the parser implementation.

SBERT was trained to decide semantic similarity between two sentences [29]. While the original SBERT model performed reasonably well in our intent capture, we also found fine-tuning the model could improve the result.

Figure 21 shows a 2D projection (using t-SNE) of the embeddings based on 3623 acceptable queries we sample.

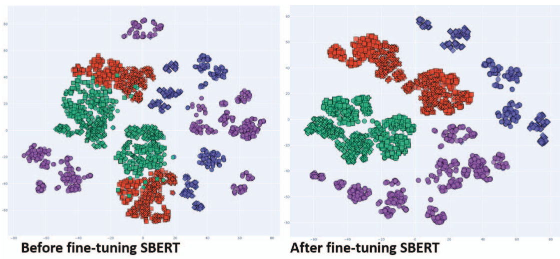

Figure 21. Fine-tuning SBERT improves our intent capture

The purple markers are queries with no context switching. The other three colors each represents switching to the particular intent. As seen, fine-tuning the model makes separation among different groups of queries more clear.

Our fine-tuning follows the *retrofitting* idea suggested in [30]. Among the 3623 queries, we sample 382 to fine-tune the SBERT (following Figure 20, this means that $n = 382$ in E_n). Then, we test the model on the remaining 3241 queries and find only 1 query whose intent is captured wrong. The cause for the mistake is actually due to the fact that paraphrasing can generate a query that looks quite different from its original query. In Figure 20, this means that we have to include this one query into the set of acceptable queries on the right to cover the special case.

6.2. Phrase matching

Our approach to phrase matching also relies on checking the cosine similarity between two embeddings. The difference is that in phrase matching, the two embeddings are from two phrases (instead of two sentences). And instead of using SBERT, we use the original BERT model [26].

Figure 22 illustrates the approach. Similar to Figure 20, we first build a table of phrase embeddings $q_1, \ldots, q_m$. Each q_j is labeled with the corresponding node name in the KG. The starting point is also a set of acceptable queries.

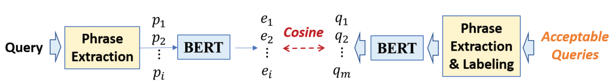

Figure 22. Phrase matching by comparing pairwise BERT embeddings

Given a query, we need to extract phrases stated in the query. We use the core-NLP API [31] to obtain a constituency parsing tree. We then applied custom rules to extract potential phrases. If the query is the original query generated by sentence completion, the label of a phrase can be determined easily by checking the phrases stored with the nodes in the KG. If the query is obtained by paraphrasing, then the label is determined by matching its BERT embedding with the BERT embeddings of those phrases from the original query.

In our current implementation we have grown the table to contain over 13K embeddings categorized with 116 labels. We have verified the performance with over 5K paraphrased queries and found no mistakes.

Given a user query, the process to obtain embeddings $e_1, \ldots, e_i$ in Figure 22 is similar. First we use core-NLP API and rules to obtain phrases $p_1, \ldots, p_i$. Then, we apply BERT to get the embeddings. After that, for each e_i we search (using cosine similarity again) the embedding table to find the best-matched q_j and then we activate the corresponding node in the KG.

Figure 23. Examples of parsing a query into a KG configuration

Figure 23 uses two query examples with two different intents to illustrate the parsing process. Note that these two queries each contains several phrases to make the parsing more difficult. In practice, most queries might be much simpler, involving simple action with one or no option (e.g. the dialog example in Figure 16).

6.3. Dialog representation of an analytic process

In typical ML, a given type of analysis (e.g. classification, clustering, etc.) is applied on a dataset to obtain a result. In contrast, we consider DSML as an iterative knowledge exploration process from the data [6][15][16]. Under this view, in each step the user explores the data from a specific *perspective*. With IEA-Plot, this perspective corresponds to the configurations implied by the queries.

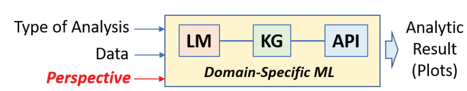

Figure 24. LM enables Domain-Specific ML (DSML) through dialog

Figure 24 depicts our DSML view. Under this view, each exploration process is represented through a dialog. Which analytic results (plots) are meaningful to a user is up to the user to decide. The DSML tool's job (IEA-Plot) is to facilitate the exploration process. LMs bring two major benefits: (1) enabling dialog representation of an analytic process; (2) improving the efficiency of the exploration.

From our DSML view, the KG provides a specification for how a particular ML tool is used within the application scope. For example, in our KG the concept of pattern classification is captured in the **Concept** node `WaferPatternFilter`. It involves seven pattern concepts. A rule-based script is currently associated with each pattern concept. Others can replace the scripts with their own. The classification relies on a Minions graph constructed based on a wafer-wafer similarity measure. Others can add new implementations with their own similarity measures. Overall, our KG separates the API implementation from its usage and provides a clear contextual definition for how a ML tool is used in the application domain.

7. Conclusion

IEA-Plot is designed to enable chat-based analytics. Inputs to IEA-Plot are queries forming a dialog. Outputs are plots. The essential idea for implementing IEA-Plot is the development and use of a KG. The main contribution of this work is showing how the KG can be constructed, which captures the domain knowledge in the specific application domain. Our KG together with the front-end parser can be shared with others, providing a platform for customizing their own IEA tool by adding their own backend API.

The performance of our frontend parser depends on the paraphrasing power of a large LM. How to make a LM better understand the terms used in our domain and improve such power is a separate research issue. While the current parser is implemented as a constrained parser, as we collect more query examples over usage, it will become feasible to consider other approaches, such as reinforcement learning [9] or joint LM+KG reasoning [19][20]. Those can be interesting future research directions.

Although IEA-Plot is designed for user to interact with the tool through a dialog, it is possible to add a separate GUI to enable using the tool's functions through mouse clicks. In this case, the focus might not be on enabling dialog-driven analytics. Instead, it can be for automatic translation of a usage session (i.e. a sequence of mouse clicks) into a natural language description (e.g. a workflow description). In IEA-Plot, each analytic step is represented as a subgraph of the KG and with the generative approach, can be converted into a natural language description. Such automatic workflow capture can be an interesting future add-on feature spanned from our current IEA-Plot design.

Acknowledgment This work is supported in part by National Science Foundation Grant No. 2006739. The authors are thankful to Sergio Mier and Leon Wang of Qualcomm for their valuable inputs to our research.

References

[1] R. Bommasani and el al., "On the opportunities and risks of foundation models," *arXiv:2108.07258*, 2021.

[2] T. B. B. et al., "Language models are few-shot learners," *CoRR (also in NeurlPS Proceedings)*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[3] M. C. et al., "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[4] L. O. et al., "Training language models to follow instructions with human feedback," *CoRR*, vol. abs/2203.02155, 2022. [Online]. Available: https://arxiv.org/abs/2203.02155

[5] OpenAI, "Introducing ChatGPT." [Online]. Available: https://openai.com/blog/chatgpt

[6] L.-C. Wang and J. Zeng, "Machine learning support for wafer-level pattern analytics," *Chapter 9 in Machine Learning Support for Fault Diagnosis of System-on-Chip, Springer Nature*, 2023.

[7] L.-C. Wang, "An autonomous system view to apply machine learning," in *IEEE International Test Conference*, 2018. [Online]. Available: https://iea.ece.ucsb.edu/iea/project

[8] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," *CoRR*, vol. abs/2201.07207, 2022. [Online]. Available: https://arxiv.org/abs/2201.07207

[9] M. A. et al., "Do as i can, not as i say: Grounding language in robotic affordances," *CoRR*, vol. abs/2204.01691, 2022. [Online]. Available: https://arxiv.org/abs/2204.01691

[10] M. J. Yang, Y. J. Zeng, and L.-C. Wang, "Language driven analytics for failure pattern feedforward and feedback," in *IEEE International Test Conference*, 2022.

[11] R. Shin and et al., "Constrained language models yield few-shot semantic parsers," *CoRR*, vol. abs/2104.08768, 2021.

[12] Y. J. Zeng, L.-C. Wang, and C. J. Shan, "Miniature interactive offset networks (minions) for wafer map classification," in *IEEE International Test Conference*, 2021, pp. 190–199.

[13] Y. J. Zeng, M. J. Yang, and L.-C. Wang, "Wafer map pattern analytics driven by natural language queries," in *IEEE International Test Conference in Asia*, 2022.

[14] A. H. et al., "Knowledge graphs," *CoRR*, vol. abs/2003.02320, 2020. [Online]. Available: https://arxiv.org/abs/2003.02320

[15] M. Nero, *Domain-Specific Machine Learning - A No-Free-Lunch Perspective*. UCSB PhD Thesis, March 2022.

[16] J. Shan, *Domain-Specific Machine Learning - A Human Learning Perspective*. UCSB PhD Thesis, March 2022.

[17] D. W. et al., "Dynamic integration of background knowledge in neural nlu systems," *CoRR*, vol. abs/1706.02596, 2017. [Online]. Available: http://arxiv.org/abs/1706.02596

[18] B. Y. Lin, X. Chen, J. Chen, and X. Ren, "KagNet: Knowledge-aware graph networks for commonsense reasoning," *CoRR*, 2019. [Online]. Available: http://arxiv.org/abs/1909.02151

[19] Y. e. a. Feng, "Scalable multi-hop relational reasoning for knowledge-aware question answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, nov 2020, pp. 1295–1309. [Online]. Available: https://aclanthology.org/2020.emnlp-main.99

[20] M. e. a. Yasunaga, "QA-GNN: Reasoning with language models and knowledge graphs for question answering."

[21] X. Z. et al., "GreaseLM: Graph reasoning enhanced language models for question answering," *CoRR*, vol. abs/2201.08860, 2022. [Online]. Available: https://arxiv.org/abs/2201.08860

[22] R. Speer, J. Chin, and C. Havasi, "ConceptNet 5.5: An open multilingual graph of general knowledge," in *AAAI Conference on Artificial Intelligence*, 2017, pp. 4444–4451. [Online]. Available: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14972

[23] A. Singhal, "Introducing the knowledge graph: things, not strings," *Google Blog*, 2012. [Online]. Available: https://www.blog.google/products/search/introducing-knowledge-graph-things-not/

[24] G. Klyne, J. J. Carroll, and B. McBridge, "RDF 1.1 Concepts and Abstract Syntax," *W3C*, 2014. [Online]. Available: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[25] P. H. et al., "OWL 2 Web Ontology Language primer (second edition)," *W3C*, 2012. [Online]. Available: https://www.w3.org/TR/2012/REC-owl2-primer-20121211/

[26] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[27] M.-J. Wu, J.-S. R. Jang, and J.-L. Chen, "Wafer map failure pattern recognition and similarity ranking for large-scale data sets," *IEEE Tran. on Semi. Manufacturing*, vol. 28, no. 1, pp. 1–12, 2015.

[28] OpenAI, "Gpt-3.5-turbo." [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5

[29] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *CoRR (also in EMNLP*, 2019. [Online]. Available: http://arxiv.org/abs/1908.10084

[30] M. e. a. Faruqui, "Retrofitting word vectors to semantic lexicons," in *Proc NA-ACL: Human Language Technologies*, 2015, pp. 1606–1615. [Online]. Available: https://aclanthology.org/N15-1184

[31] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, "Stanza: A Python natural language processing toolkit for many human languages," in *Proc. 58th ACL: System Demonstrations*, 2020. [Online]. Available: https://nlp.stanford.edu/pubs/qi2020stanza.pdf