Wafer Map Pattern Analytics Driven By Natural Language Queries

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

***Abstract*—We present a novel approach where wafer map pattern analytics are driven by natural language queries. At the core is a semantic parser that translates a user query into a meaning representation comprising instructions to generate a summary plot. The allowable plot types are pre-defined which serve as an interface that communicates user intents to the analytics software backend. Application results on wafer maps from a recent production line are presented to explain the capabilities and benefits of the proposed approach.**

1. Introduction

Wafer Map Pattern Recognition (WMPR) is a problem that has been studied for decades in the field of semiconductor manufacturing. A notable dataset was developed in [1], called WM-811K. This dataset includes 811,457 wafer maps where 172,950 are with labels and the rest without. The work [1] presented a comprehensive learning approach based on engineering a set of *features* to encode each wafer map into a vector of values, to be learned by a learning tool such as SVM [2]. Many works were published later based on the dataset [1][3][4][5], including those more recent works involving deep learning techniques [6][7][8][9][10].

The WM-811K dataset pre-defines a number of *classes* to differentiate wafer maps [1]. In addition to eight classes to differentiate wafer maps having a "pattern", a "None" class is used to indicate "no pattern". Given such a dataset, it is natural to treat WMPR as a multi-class classification problem, where the focus is on optimizing the accuracy of a classifier (a SVM model or a neural network model).

While optimizing model accuracy might be a common practice from the perspective of a ML (machine learning) technology developer, it is not always intuitive from a test practitioner's point of view. For example, in addition to knowing that a set of wafer maps all exhibiting the "same" pattern, a practitioner might also desire to know if some action can be taken based on the pattern.

Figure 1 depicts a context of feedforward from wafer probe to final test. Suppose three wafer maps are deemed to show a systematic pattern. For the practitioner, knowing this is not enough. Further, the practitioner would like to know if there is some correlation to the final test result. An intuitive way is to plot a *stacked wafer map* based on all failing parts from final test. Then, between the three wafer maps and the stacked wafer map, the practitioner can observe that the final test fails tend to form an "inner ring" next to the "outer ring" shown on the wafer maps. This proximity relationship can be considered as a form of correlation where its final interpretation and utilization can then be subjected to further investigation and discussion from a team.

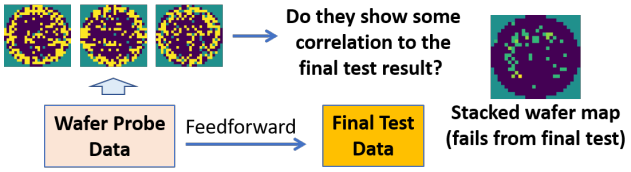


Figure 1. A feedforward context from wafer probe to final test

In the context of Figure 1, getting to an analytic result involves three steps: (1) obtain a set of wafer maps showing some systematic pattern; (2) plot the stacked wafer map; (3) decide if a correlation exists. Step 1 is key to the process. Once the wafer maps and the stacked wafer map are displayed, a person can judge if a correlation exists.

In this context, treating step 1 as solving a multi-class classification problem might not be effective [11]. This is because the wafer maps selected in the set affect how the stacked wafer map looks like and consequently, can dictate if a correlation shows up. Automating step 3 might not be necessary, nor effective either. Once the plots are displayed, a person can see a correlation visually. If there is non-visualizable correlation the person would like to check, the person can always write a separate software script to do it.

Because searching for the right set of wafer maps is the essence of the problem, one can think of a software App that helps a practitioner to search for a result like that shown in Figure 1. The search essentially follows a repeated process of "try-and-see". In each try, a set of wafer maps is selected based on a user intent and the correlation is decided by the user from visualizing the output plots.

The work presented in this paper is for implementing such a software App. We envision that the user interacts with the App through a natural language interface (much like how they would interact with a *virtual assistance*). This work focuses on building the interface to drive the analytics backend presented in [11]. At the core of the work is a novel *semantic parser* that translates user queries into instructions to the analytics backend.

The rest of the paper is organized as the following. Section 2 provides a high-level picture for realizing the App. Section 3 presents the key ideas of the approach. Section 4 describes details of our semantic parser implementation. Section 5 discusses interesting findings based on test data from a recent production line. Section 6 concludes.

2. Natural Language Driven Analytics

Figure 2 depicts the major components in our approach. It comprises a natural language frontend and an analytics backend, connected through a *plot-based worldview*. The worldview defines what types of plot can be produced as well as for each plot, what *plot attributes* can be controlled.

Figure 2. Major components in our approach

The worldview essentially defines an actionable space for the frontend. The core of the frontend is a *semantic parser*. The parser translates a user's input in natural language into a sequence of instructions which perform two tasks: (1) Data Collection: determine the *set of wafer maps* that is used for generating a plot; (2) Plotting: select a type of plot and set the values of the plot attributes.

The backend supports two types of analytics: Minions' analytics and NLI's analytics (NLI stands for Natural Language Interpreter). The two analytics components are based on the techniques reported in two of our prior works [11][12]. While they provide two different perspectives to analyze the wafer maps, the NLI's analytics is carried out on top of the Minions' analytics [11].

From the high-level view, each perspective provides a set of *wafer attributes*. For each wafer map, the two analytics components determine the values of those attributes. In a sense, one can think that analytics results are stored as a database table where each row corresponds to a wafer map and each column corresponds to a wafer attribute.

TABLE 1. A SNAPSHOT OF THE WAFER MAP DATABASE TABLE

Wafer	Meta Attr.		NLI Attribites					Minions Attr.	
	stage	...	id	shape	region	direction	...	group id	...
	wafer probe	...	1	arc	edge	12	...	1	...
			2	cluster	center	lower right	...		
			...						
	wafer probe	...	1	arc	edge	12	...	1	...
			2	cluster	center	upper left	...		
			...						
	wafer probe	...	1	arc	edge	2–7	...	2	...
			...						
			...						

Table 1 shows a snapshot of such a table. For example, a query might request to retrieve all wafer maps from wafer probe that have "arc along edge at 12 o'clock". The first two wafer maps will be included in the return. The "group id" attribute is provided by the Minions component [12]. The wafer maps annotated by the same group id generally exhibit a very similar pattern [11]. If the query requests "group 1", the result will also include the first two wafer maps.

In summary, our semantic parser should support two categories of queries, one to control the plotting and the other to control the wafer map set retrieved from the database.

3. Implementation Of A Semantic Parser

Semantic parsing is the process to assign real-world meanings to linguistic inputs [13] (e.g. words). Specifically, in *computational semantics*, formal structures called *meaning representations* are used to link the non-linguistic knowledge (e.g. data stored in database, API function calls, etc.) to linguistic elements such as English words. Semantic parsing is a wide field of study. In machine learning, most works to achieve semantic parsing centered on using supervised learning with large amounts of human-created semantic parses [13][14]. Such an approach is hard to be duplicated in a specialized domain like ours, because we lack the resources to create large amounts of training data.

Consequently, we do not take a supervised learning approach to implement our parser. Instead, we adopt the approach called *semantic parsing as paraphrasing* [15]. The approach makes use of triples (*natural query* q, *canonical utterance* c, *meaning representation* m), where the parser maps $q \rightarrow c \rightarrow m$. It was observed in [15] that mapping $c \rightarrow m$ and vice-versa could be achieved by fixed rules, which make it more feasible in our application context.

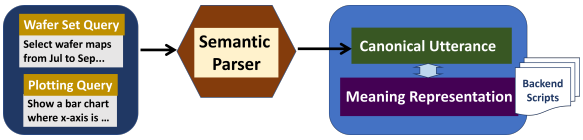


Figure 3. A semantic parser interfacing with natural language queries

Figure 3 depicts the idea in our context. As mentioned above, two types of queries are supported: wafer set query and plotting query. Then, the semantic parser translates queries into *canonical utterances* which are based on a restricted language whose *lexicon* is custom-defined by us. These utterances are interpreted with a *meaning representation* which defines the corresponding actions to be performed by the backend analytics component. In this flow, the "$c \rightarrow m$" mapping are implemented with fixed rules. The "$q \rightarrow c$" mapping are handled by the parser.

Realizing Figure 3 requires us to define a feasible meaning representation as the target for the parser. We can start by developing a *grammar* that defines the lexicon of the canonical utterances. After the lexicon and the meaning representation are defined, we can choose a way to implement the parser. In our work, the parser is implemented with *constrained semantic parsing* powered by a pretrained language model [16]. Below, we will first discuss the meaning representation and the grammar. The constrained semantic parsing will be discussed in Section 4.

3.1. Plot-based worldview

Our plot-based worldview defines a collection of plot types each with some plot attributes. Below, we use a particular plot example to illustrate this idea.

3.1.1. A summary plot example. Recall the feedforward context discussed in Figure 1. Considering what type of summary plot might be useful in the context, Figure 4 shows an example plot type. The wafer maps used in the plot were from a recent production line. The wafers were stamped with days in four months (from October to January). The four months had 211, 1501, 1337, and 3251 wafers, respectively.

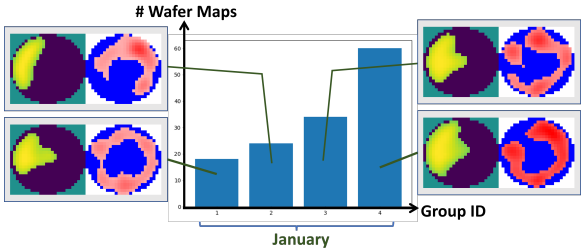


Figure 4. An example plot type in the feedforward context

The plot shows that in January, the analytics found four groups of wafers (based on "group id" attribute shown in Table 1). The bar height shows the number of wafers in each group. For each group, two wafer images are shown. The left image is obtained by stacking wafer maps in the group where those wafer maps are from the wafer probe stage. The right image is by stacking wafer maps where they are based on the final test. Each stacked image shows the smallest region including 60% of the density, i.e. the region including roughly 60% of the fails in the group. In the rest of this paper, we refer to the two stacked images as *wafer probe heatmap* and *final test heatmap*, respectively.

The correlation between wafer probe and final test can be inspected in this summary plot. It can be observed that the failing pattern forms a "thick ring". Fails from wafer probe concentrate on the left-side of the ring. Then, fails from final test tends to "complete" the ring.

It is important to note that Figure 4 is generated based on user queries. Hence, user needs to specify what set of wafer maps they want to inspect in a plot. In the following, we will use the example in Figure 4 to illustrate how our frontend can take user queries and generate such a plot.

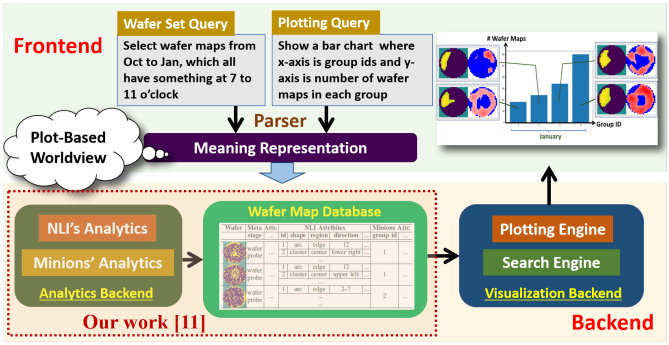


Figure 5. The workflow from queries to a summary plot

3.1.2. From queries to summary plot. Figure 5 shows the workflow from queries to a summary plot. The meaning representation contains instructions in the backend for retrieving a set of wafers from the wafermap database. The software scripts executing these instructions are implemented in the search engine module. In addition, there are instructions for generating a certain type of plot and assigning plot attributes based on the retrieved data. The corresponding software scripts are implemented in the plotting engine module. In view of our prior work [11], the frontend and the two engines in the backend are new.

3.1.3. Grammar and lexicon. To define the meaning representation, we define the following: (1) A grammar that captures the data gathering and plotting operations in the backend. (2) From the grammar, we obtain the lexicon for defining the meaning representation.

TABLE 2. A SNIPPET OF THE GRAMMAR AND LEXICON

Grammar Rules	
(i)	$S \rightarrow Plot$
(ii)	$Plot \rightarrow Bar \mid Scatter \mid Wafermap \mid ...$
(iii)	$Bar \rightarrow Descr_{Bar}$ TYPEBAR
(iv)	$Descr_{Bar} \rightarrow$ AXIS1 AXIS2 TITLE
	...

Lexicon
TYPEBAR → bar chart
TITLE → title | *null*
AXIS1 → x-axis
AXIS2 → y-axis
...

For defining the grammar, we follow an approach similar to that used in [11]. We define a context-free grammar that describes the process for generating a plot. For example, the left-hand side of Table 2 shows a snippet of the grammar, which consists of a set of production rules. Rule (i) starts the process to get a plot. Rule (ii) provides several plot options including *Bar*, *Scatter*, or *Wafermap* etc. In Rule (iii), each plot option defines the plot type and a descriptor for the specific type. For bar chart, the descriptor specifies three attributes: AXIS1, AXIS2, and TITLE.

Production rules which derive words as the terminal values are called the lexicon. A snippet of the lexicon is shown on the right-hand side of Table 2. The left part of a lexicon rule has a type or attribute that can be processed internally by the backend. For example, *TypeBar* corresponds to a script in the plotting engine to define a bar plot object. The lexicon rules provide the words that can be used in the utterances and in the meaning representation. For example, the plotting query in Figure 5 uses words "bar chart", "x-axis", and "y-axis". Note that the lexicon related to wafer set queries is developed in the work [11].

TABLE 3. A SNIPPET OF OPERATORS AND THEIR EXPRESSIONS

Operator	Template	Meaning Representation
Select	Return [TypeData]	**1. Return wafer maps**
Filter	Return [ref] from [condition]	1. Return wafer maps **2. Return #1 from Jul to Sep**
Aggregate	Return [aggregate] of [ref]	1. Return wafer maps **2. Return the number of #1**
Show	Return [TypePlot]	**1. Return bar chart**
Group	Return [aggregate] [ref1] for each [ref2]	1. Return wafer maps 2. Return group ids of #1 **3. Return the number of #1 for each #2**
Specify	Return [ref1] where [plot attribute] is [ref2]	1. Return bar chart 2. Return wafer maps. 3. Return group ids of #2 **4. Return #1 where x-axis is #3**
	...	

3.1.4. Meaning representation and its operators. The meaning representation uses words provided by the lexicon rules. In addition, it has words that express operations. For example, to generate the plot in Figure 5, the query can use the word "select" to request a set of wafer maps from the database. Then, a separate query uses the word "show" to request a plot object. Table 3 shows a snippet of these so-called *operators*. For each operator, the natural language

template used to express the operator is shown. The parameters denoted by "[]" are from the lexicon. The words outside are called *functional words* which indicate what functions to be performed on the parameters. The corresponding meaning representation exemplifies these functions.

This meaning representation is called *Question Decomposition Meaning Representation* (QDMR) [17], chosen for our current implementation. QDMR contains natural utterances, which is easier to understand than complex logic forms used in [14]. Using QDMR facilitates data preparation for us to bootstrap a semantic parser.

In general, QDMR constitutes an ordered list of steps, each corresponds to an operator. In our case, each operator can be executed by some backend scripts. The steps all together accomplish the task specified in the user query. The lexicon defined by our grammar restricts the scope of meaning representations to those supported by the backend. A valid meaning representation should contain words only from the pre-defined lexicon, the functional words used in the operator template such as "for each", and possibly a *reference token* that refers to the result of a previous step. In this way, we make sure that any step in the meaning representation can be mapped into a valid execution flow with the backend scripts. Moreover, some operators can be merged to a *high level* QDMR [17] to reduce the total number of steps involved. The detailed definition of QDMR and more types of operators can be found in [17].

4. Constrained Semantic Parsing

For semantic parsing, we adopt the method proposed in [16], illustrated in Figure 6. The method leverages a pretrained language model (LM) for parsing input queries into canonical utterances. With a LM, we can use *few-shot learning* to teach the LM about our specific utterances.

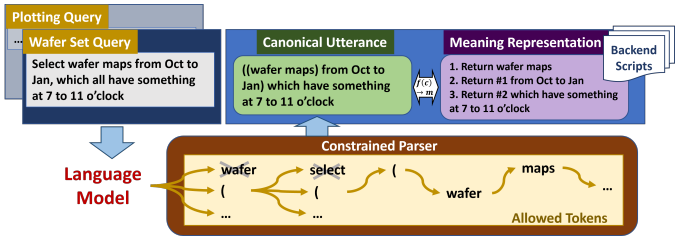


Figure 6. Constrained parsing flow

Following [16], we use the state-of-the-art language model GPT-3 [18] that has 175 billion parameters as the natural language interface. The GPT-3 handles the variation of wordings in the spoken language. Moreover, by utilizing GPT-3's powerful *in-context learning*, we finetune the GPT-3 to learn the *translation* from the input natural query to its corresponding canonical utterance with only 20 demonstrating examples (i.e. few-shot learning). The translation is essentially a many-to-one mapping function that generates a single unique interpretation of various sayings.

As shown in Figure 6, the constrained parsing restricts the output space of GPT-3 by a set of *allowed tokens*, i.e. words that should be generated by GPT-3 based on the input query. The allowed tokens include: (i) words or their inflections that appear in the query (words from the lexicon), (ii) a pre-defined set of functional words (words used in operator's template), (iii) opening and closing parentheses.

Given a natural query as input, GPT-3 will search through all valid tokens that belong to one of these three categories and output the token that has the highest probability conditioned on the tokens already generated. In addition, the parser will ensure any parentheses used in the output string are balanced. The generated string by GPT-3 becomes our canonical utterance for the query.

Then, it is straightforward to convert a canonical utterance into a meaning representation in QDMR format. For example, a converting function $f(c) \rightarrow m$ can be implemented by: for each utterance in the parentheses, replacing the return in inner parentheses with reference token, removing all the parentheses, and adding the word "return" to the front and new line to the end. Each meaning representation entails an executable flow in our backend to generate a plot.

4.1. Parsing examples

To illustrate how the parsing works, Table 4 shows two example queries. The GPT-3 parses the query into a canonical utterance which is then converted into the meaning representation. In the utterance for Query 1, the words "wafer maps", "wafer probe", "bin", "component yield loss" are tokens from the lexicon. The words "from", "with", "greater than" are functional words from the template.

TABLE 4. PARSING EXAMPLES INTO $\mathcal{C}$ AND $\mathcal{M}$

Query 1	*Give me wafer maps from wafer probe from bin $\mathcal{Z}$ with component yield loss greater than 10 percent*
(((wafer maps) from wafer probe) from bin $\mathcal{Z}$) with component yield loss greater than 10 percent	
1. Return wafer maps 2. Return #1 from wafer probe 3. Return #2 from bin $\mathcal{Z}$ 4. Return #3 with component yield loss greater than 10 percent	
Query 2	*Show a bar chart where x-axis is group ids, y-axis is the number of wafers in each group*
((bar chart) where x-axis is group ids) where y-axis is number of wafers for each group	
1. Return bar chart 2. Return #1 where x-axis is group ids 3. Return #2 where y-axis is number of wafers maps for each group id	

As discussed before, these tokens restrict the scope of GPT-3's output. Since the utterance is well-structured, converting it to the list of operations in the meaning representation is straightforward. While Query 1 defines a set of wafer maps that can be used in a plot, Query 2 defines the plot type and attributes for generating a plot.

4.2. Generating Figure 4

Table 5 below shows the two queries, their parsed canonical utterances, and the mean representations, specifically used for generating the plot shown in Figure 4 before.

It should be noted that the utterance "something left along edge" describes a certain type of wafer map pattern the query is looking for. The set of wafer maps with such a pattern will be retrieved from the database. This part of the functionality is developed in our work [11]. The returned

TABLE 5. QUERIES THAT GENERATE FIGURE 4 SHOWN BEFORE

Wafer Set Query
*Select wafer maps from wafer probe from bin $\mathcal{Z}$ with component yield loss greater than or equal to 5% from Oct to Jan, where all wafer maps exhibit something **left** along the edge*
(((((wafer maps) from wafer probe) from bin $\mathcal{Z}$) with component yield loss greater than or equal to 5%) from Oct to Jan) which have something left along edge
A.1. Return wafer maps
A.2. Return A.1 from wafer probe
A.3. Return A.2 from bin $\mathcal{Z}$
A.4. Return A.3 with component yield loss greater than or equal to 5%
A.5. Return A.4 from Oct to Jan
A.6. Return A.5 which have something left along edge
Plotting Query
Show a bar chart where x-axis is group ids, y-axis is the number of wafers in each group. For each bar, also show a sub plot of two wafer maps where the left is wafer probe heatmap, the right is final test heatmap
(((bar chart) where x-axis is (group id of wafer maps)) where y-axis is (the number of wafer maps for each (group id of wafer maps))) with (((wafermap subplot for each bar) where left figure is wafer probe heatmap) right figure is final test heatmap)
B.1. Return bar chart
B.2. Return group id of A.1
B.3. Return the number of A.1 for each B.2
B.4. Return B.1 where x-axis is B.2
B.5. Return B.4 where y-axis is B.3
B.6. Return B.5 with wafer map subplot for each bar
B.7. Return B.6 where left is wafer probe heatmap
B.8. Return B.7 where right is final test heatmap
Generated Plot
See Figure 4 shown before

set will be organized in *groups* by default and the grouping is based on the Minions' analytics mentioned earlier [11].

In this example, the plotting query specifies the plot type as a bar chart, where each bar is associated with two wafer maps, the wafer probe heatmap and the final test heatmap. The plot attributes are also given as x-axis is the *group ids*, and the y-axis is *the number of wafer maps in each group*.

Note that our approach enforces the ordering that wafer set query (A) must be made before plotting query (B). Because some attributes from B needs to be determined based on the results returned by A. For example in Table 5, to accomplish the step B.2 in meaning representation of B requires the return from step A.1 of A.

5. Key Findings In Application

As mentioned in Section 3.1.1, we applied our approach to analyze four months of wafer maps from a recent production line. This "analysis" was essentially a query-driven "try-and-see" search process. In each try, we query for a plot and inspect the result. In the process, we collected the plots that to us, represent a potentially interesting finding.

It should be noted that in our search for the interesting plots, a set of wafer maps is primarily defined by the "pattern" we are looking for. Usually, a wafer map is based on all fails on the wafer. For this dataset, though, we found that only using all-fail to draw wafer maps is not sufficient. In addition to using all-fail, we also had to search on wafer maps based on fails from individual *test bins*. Figure 7 uses two examples to illustrate the need to search on individual test bins. On each example, the left plot shows the wafer map based on all fails. The right shows two wafer maps, each based on a particular test bin. As seen, wafer maps based on a test bin can more clearly exhibit a pattern.

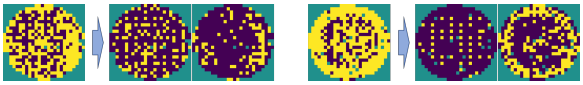


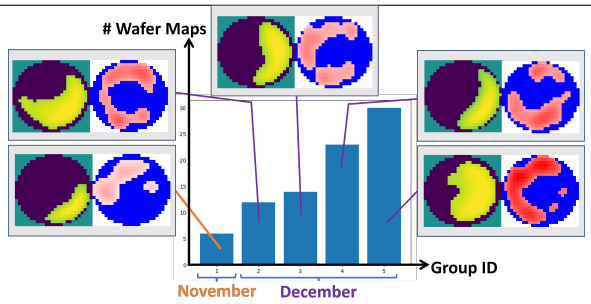
Figure 7. Examples of wafer map decomposition based on test bins

In our analytics backend, each wafer has multiple wafer maps defined, based on total fails and based on a set of defined test bins. In a try, if no test bin is specified, all these wafer maps are considered. For example, the two wafer maps with a "grid pattern" shown in Figure 7 could be from two different test bins, and when a query simply requests a "grid pattern", they both could be put into a single set.

5.1. Finding #1

Continuing the result shown in Figure 4, Table 6 shows a result that also indicates a "thick ring" fail pattern. For those wafer probe heatmaps, the patterns are on the right side (in contrast to left side in Figure 4). However, when we look at both wafer probe heatmaps and final test heatmaps, we can see, together all pairs also tend to form a "thick ring" pattern (like that shown before in Figure 4).

TABLE 6. SECOND RESULT

Wafer Set Query
Get wafer maps from bin $\mathcal{Z}$ of wafer probe with yield greater than or equal to *5% from Oct to Jan, which all have something **right** along the edge*
(((((wafer maps) from wafer probe) from bin $\mathcal{Z}$) with component yield loss greater than or equal to 5%) from Oct to Jan) which have something right along edge
Plotting Query
Plot a bar chart such that x-axis is group ids, y-axis is the number of wafers in each group. Also show a sub plot of two wafer maps for each bar, where the left is wafer probe heatmap, the right is final test heatmap
(((bar chart) where x-axis is (group id of wafer maps)) where y-axis is (the number of wafer maps for each (group id of wafer maps))) with (((wafermap subplot for each bar) where left figure is wafer probe heatmap) right figure is final test heatmap)
Generated Plot


Then, we combine the two plots with another plot (not shown) which has only 1 wafer group, to produce a final summary plot shown in Figure 8. This plot facilitates one to visualize that the "thick ring" fail pattern trend appeared on many wafers and lasted from November to Janurary.

5.2. Finding #2

Table 7 shows another interesting finding we discovered on the dataset, by searching for a "grid pattern". The plot

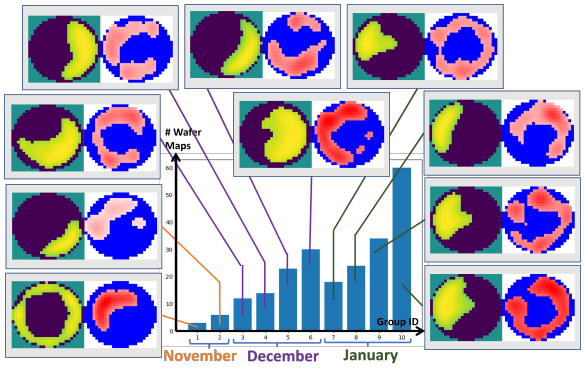
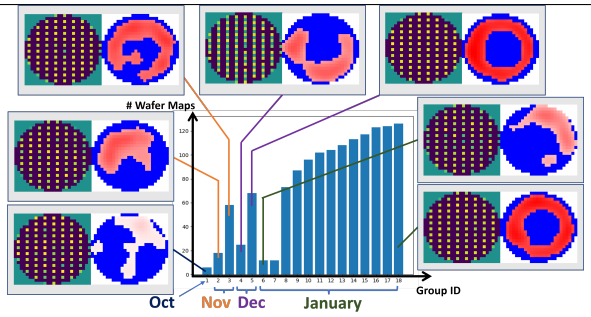


Figure 8. Merged summary plot of Table 5 and Table 6

shows that many wafers were affected (with this pattern) at wafer probe stage, lasting from October to Janurary. However, there is no apparent correlation to the final test. The underlying cause requires further investigation beyond the scope of the analytics provided by the App.

TABLE 7. THIRD RESULT

Wafer Set Query
*Fetch wafer maps from wafer probe with component yield loss greater than or equal to 5% from Oct to Jan, which all demonstrate a **grid** pattern*
((((wafer maps) from wafer probe) with component yield loss greater than or equal to 5%) from Oct to Jan) which have grid
Plotting Query
Generate a bar chart where y-axis is the number of wafers in each group, x-axis is group ids. And near each bar, also draw a sub plot of two wafer maps the right is final test heatmap, the left is wafer probe heatmap.
(((bar chart) where x-axis is (group id of wafer maps)) where y-axis is (the number of wafer maps for each (group id of wafer maps))) with (((wafermap subplot for each bar) where left figure is wafer probe heatmap) right figure is final test heatmap)
Generated Plot


6. Conclusion

In this work, we present a novel approach where analytics of wafer map patterns are driven by natural language queries. The proposed approach enables a user to conduct a search of interesting analytic results at natural language level, which is more intuitive and more efficient. The analytic results are shown as plots to facilitate a user to decide if they are interesting or not. The core of this work is implementing a semantic parser. Our current implementation is based on the meaning representation QDMR [17]. There are other choices for this meaning representation, which can be explored further to provide more capabilities to our frontend. Our current parser relies on GPT-3. There are other language models which can be experimented with. This aspect will be studied in future work.

Acknowledgment This work is supported in part by National Science Foundation Grant No. 2006739 and by Semiconductor Research Corporation project No. 2020-CT-2933. The authors are thankful to Sergio Mier and Leon Wang of Qualcomm for their valuable inputs to our research.

References

[1] 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.

[2] B. Schölkopf and et al., *Learning with Kernels:Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[3] M. Fan, Q. Wang, and B. van der Waal, "Wafer defect patterns recognition based on optics and multi-label classification," *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2016.

[4] J. Yu and X. Lu, "Wafer map defect detection and recognition using joint local and nonlocal linear discriminant analysis," *IEEE Tran. on Semi. Manufacturing*, vol. 29, no. 1, pp. 33–43, 2016.

[5] a. a. Minghao Piao, "Decision tree ensemble-based wafer map failure pattern recognition based on radon transform-based features," *IEEE Tran. on Semi. Manufacturing*, vol. 31, no. 2, pp. 250–257, 2018.

[6] J. Yu, "Enhanced stacked denoising autoencoder-based feature learning for recognition of wafer map defects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 613–624, 2019.

[7] N. Yu, Q. Xu, and H. Wang, "Wafer defect pattern recognition and analysis based on convolutional neural network," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 566–573, 2019.

[8] J. Wang, Z. Yang, J. Zhang, Q. Zhang, and W.-T. K. Chien, "Adabalgan: An improved generative adversarial network with imbalanced learning for wafer defective pattern recognition," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 3, pp. 310–319, 2019.

[9] T.-H. Tsai and Y.-C. Lee, "A light-weight neural network for wafer map classification based on data augmentation," *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 4, pp. 663–672, 2020.

[10] M. Saqlain, Q. Abbas, and J. Y. Lee, "A deep convolutional neural network for wafer defect identification on an imbalanced dataset in semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 3, pp. 436–444, 2020.

[11] 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.

[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] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 3rd ed., 2022.

[14] Y. Wang and et al., "Building a semantic parser overnight," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, 2015.

[15] J. Berant and P. Liang, "Semantic parsing via paraphrasing," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2014.

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

[17] T. Wolfson and et al., "Break it down: A question understanding benchmark," *CoRR*, vol. abs/2001.11770, 2020.

[18] T. Brown and et al., "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020.