# What Is Wrong with My Model? Identifying Systematic Problems with Semantic Data Slicing

Chenyang Yang
Carnegie Mellon University

Yining Hong
Carnegie Mellon University

Grace A. Lewis
Carnegie Mellon Software Engineering Institute

Tongshuang Wu
Carnegie Mellon University

Christian Kästner
Carnegie Mellon University

## ABSTRACT

Machine learning models make mistakes, yet sometimes it is difficult to identify the systematic problems behind the mistakes. Practitioners engage in various activities, including error analysis, testing, auditing, and red-teaming, to form hypotheses of what can go (or has gone) wrong with their models. To validate these hypotheses, practitioners employ data slicing to identify relevant examples. However, traditional data slicing is limited by available features and programmatic slicing functions. In this work, we propose SemSlicer, a framework that supports semantic data slicing, which identifies a semantically coherent slice, without the need for existing features. SemSlicer uses Large Language Models to annotate datasets and generate slices from any user-defined slicing criteria. We show that SemSlicer generates accurate slices with low cost, allows flexible trade-offs between different design dimensions, reliably identifies under-performing data slices, and helps practitioners identify useful data slices that reflect systematic problems.

**ACM Reference Format:**
Chenyang Yang, Yining Hong, Grace A. Lewis, Tongshuang Wu, and Christian Kästner. 2024. What Is Wrong with My Model? Identifying Systematic Problems with Semantic Data Slicing. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3691620.3695033

## 1 INTRODUCTION

Machine learning models exhibit undesired behaviors, yet it is often hard to identify systematic problems behind individual errors. Models have been found to perform worse on under-represented subgroups [28], exhibit unintended biases [46], and produce harmful content [55], which can cause project failures, media controversies, and even lawsuits once they are integrated into software products [17]. Academia and industry have spent significant effort to help identify these problems, through activities such as error analysis, testing, auditing, and red-teaming [15, 35, 36, 43]. All these activities (Figure 1, top) curate *individual errors*, but more importantly, also create concrete *hypotheses* about the underlying systematic problem.[1]

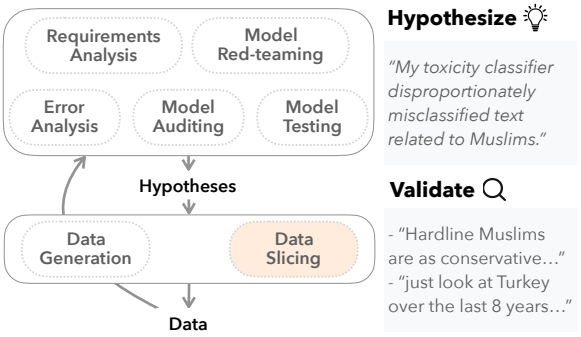


**Figure 1: ML model quality assurance involves two stages: (1) Hypothesize and (2) Validate. Many activities focus on creating hypotheses, either explicitly (requirements analysis, error analysis) or implicitly in the process (testing, auditing, red-teaming). Data slicing helps validate the produced hypotheses by identifying *additional* relevant examples, often from evaluation and production data.**

As our running example, suppose an ML practitioner is conducting error analysis on their toxicity classifier [22]. The practitioner observed that in a few examples, the model classifies non-toxic text mentioning Muslims as toxic. From these individual errors, they hypothesize that the systematic problem behind the errors might be that *"the classifier disproportionately misclassified text related to Muslims."* To understand whether their hypotheses hold, the practitioner needs to *validate* the hypotheses on more data points (Figure 1, bottom).

To validate their hypotheses, developers need to (a) conduct synthetic data generation [21] to create more data points or (b) identify relevant data points from existing data. *Data slicing* [39, 56], an example of the latter type of technique, identifies a subset of examples sharing common characteristics from existing data. Data slicing often assumes access to existing relevant features, which might not always be available for users' slicing criteria of interest. For example, it is unlikely that the input data is readily labeled with whether they are related to Muslims.

In practice, developers try to create additional features to augment datasets for data slicing but are limited by *how* they can create such features: Existing practices mostly apply *programmatic* slicing (see statistics in Table 1), often implemented as simple Python programs, to identify slices. For our example hypothesis (*"the classifier*

[1] Alternatively, the hypotheses can also be formed top-down through explicit **requirements analysis** [e.g., 5, 59], which is less common in current ML practice.


*ASE '24, October 27-November 1, 2024, Sacramento, CA, USA*

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695033

*disproportionately misclassified text related to Muslim"*), a simple implementation is to use a regular expression to search for the pattern "muslim|islam" (Figure 2, top). However, this simple implementation leaves out many related but more nuanced examples (e.g., when the examples mention "Saudi Arabia", "Turkey", as shown in Figure 2, bottom). Writing a regex-based slicing function to cover all these different cases can be a laborious process, and it is hard to enumerate these patterns to begin with. As we will show later in Section 2, this is a common theme in data slicing—developers often want to slice based on criteria that are hard to implement with programs. Limitations of existing *programmatic* slicing methods fundamentally constrain the applicability of data slicing.

In this work, we propose SemSlicer, a framework that supports semantic data slicing using Large Language Models (LLMs). *Semantic data slicing* identifies a semantically coherent subset of examples, without the need for existing features as in programmatic data slicing. The key insight of SemSlicer is that with appropriate prompts, LLMs can act as slicing functions for any user-specified slicing criteria. These slicing functions formed by an LLM and a prompt are able to cover semantically coherent examples of different surface patterns. To assist users in creating slicing prompts, we design a highly configurable prompt construction pipeline in SemSlicer, which allows different levels of machine and human intervention.

To use SemSlicer, a user needs to specify (a) a slicing criterion, expressed as a keyword or phrase (e.g., "Muslim"), (b) a dataset to slice (e.g., a test dataset or recent production data), and (c) the exact configuration for slicing (i.e., choosing among various forms of auto-optimization to trade off cost and accuracy, as we will explain in Section 3). SemSlicer will then (1) generate and automatically optimize a slicing prompt (e.g., generating few-shot examples), and (2) annotate the entire dataset and return a relevant slice. Applying SemSlicer to our running example (Figure 2, bottom), the slicing criterion can be as simple as "Muslim" (line 14), from which SemSlicer will produce a slicing prompt (line 23) that can be used to obtain the corresponding slice (line 26).

Besides model debugging as in our example, semantic data slicing is widely applicable to many activities in ML engineering, such as assisted data curation, systematic model evaluation against requirements, and model monitoring (as we will discuss in Section 2). To accommodate different use cases, SemSlicer provides a rich set of configuration options, such that users can make flexible trade-offs: adjust the models used in different components according to compute budgets; change whether and how to produce few-shot examples to trade off between compute resources and accuracy; and collaborate with LLMs in producing instructions, to provide additional human oversight. Our evaluation shows that SemSlicer can produce accurate slicing functions for a wide range of slicing criteria, allows flexible trade-offs between cost and accuracy, and is useful for model evaluation (Section 4).

In summary, our work makes the following contributions:

- A comprehensive view of the landscape of data slicing in ML engineering.
- A highly configurable framework, SemSlicer, that supports semantic data slicing for diverse use cases.[2]

[2]SemSlicer is available open-source at https://github.com/malusamayo/SemSlicer.

```
1  # Programmatic data slicing detects texts with simple
       patterns but generalizes poorly.
2  import re
3  data = load_training_data()
4  pattern = r"muslim|islam"
5
6  def regex_slicing(x: str) -> bool:
7      return bool(re.search(pattern, x, re.IGNORECASE))
8
9  m_slice = data[data['text'].map(regex_slicing)]
10 m_slice['text'].sample(2)
   | '''there's the muslim lady in jail for trying to kill
   |      people with a golf club in CTC
   |    Europe is in the process of submitting to Islam a
   |      bit more every year.'''
```

```
11 # Semantic data slicing detects texts with different
       surface patterns.
12 from semslicer.slicer import InteractiveSlicer
13 data = load_training_data()
14 criterion = "Muslim"
15
16 slicer = InteractiveSlicer(criterion, data, config={
17         'few-shot': True,
18         'few-shot-size': 8,
19         'instruction-source': 'template',
20         'student-model': 'flan-t5-xxl',
21         'teacher-model': 'gpt-4-turbo-preview'})
22
23 slicer.show_prompt()
   | '''Is the text related to muslim?
   |
   | Text: Hardline Muslims are as conservative ...
   | Answer: yes
   |
   | Text: In fact, Christmas is a pagan festival ...
   | Answer: no
   |
   | ...
   |
   | Text: {text}
   | Answer: '''

24 llm_slicing = slicer.gen_slicing_func() # str -> bool
25 m_slice = data[data['text'].map(llm_slicing)]
26 m_slice['text'].sample(2)
   | '''The solution to our problem is becoming much more
   |      stricter on immigration, banning those with
   |      religious alliances to Saudi Arabia and its
   |      puppet states.
   |    just look at Turkey over the last 8 years, the
   |      regression under the AKP speaks for itself.'''
```

**Figure 2: Existing practices mostly apply *programmatic* data slicing (cf. Table 1). For our running example, we can use a simple regex to detect comments with the phrase "muslim" or "islam" (line 6). In contrast, SemSlicer supports *semantic* data slicing, by using LLM and generated prompts as slicing functions for any user-provided criteria (line 24). The examples here are from the CivilComments dataset [7]—they do not represent the authors' view.**

- An extensive evaluation of SemSlicer that shows it can generate *accurate* slicing functions, allows *flexible* trade-offs, and is *useful* for model evaluation.

## 2 DATA SLICING

*Data slicing in ML engineering.* ML engineering is data-centric. ML practitioners usually start with data curation to obtain appropriate training and evaluation data; with LLMs, practitioners may need less data but still need data to develop and test their prompts [63].

| Slicing Criteria | Description | Examples | Frequency | Mechanism |
|---|---|---|---|---|
| domain | The domain of the text | virology, philosophy | 65 | Existing annotations |
| task | The task of the text | data understanding | 27 | Existing annotations |
| language | The language of the text | french, tamil | 20 | Existing annotations |
| script | The script of the text | latin, arabic | 6 | Existing annotations |
| site | The websites an agent needs to access | gitlab, reddit | 6 | Existing annotations |
| string_length | The length of the text (input/output/label) | - | 61 | Simple computation |
| num_steps | The number of steps an agent takes | - | 10 | Simple computation |
| num_range | The range of numerical answers | - | 16 | Simple computation |
| num_of_choices | The number of choices for multi-choice questions | - | 11 | Simple computation |
| pred_change_after_norm | Whether predictions change after normalization | - | 8 | Simple computation |
| word_repetition | The max number of word repetition | - | 3 | Simple computation |
| output_shape | The shape of model output | Yes/No, Valid/Invalid | 15 | Regex matching |
| input_type | The type of input instruction/question | what, how | 11 | Regex matching |
| is_X_library_used | Whether the input program uses library X | pandas, numpy | 11 | Regex matching |
| output_value | The value of model output | A, B, no enough info | 5 | Regex matching |
| has_function_calls | Whether the input program contains function calls | - | 4 | Regex matching |
| topic | The topic of the text | government | 3 | Machine learning (topic modeling) |

**Table 1: We analyzed 20 most popular projects on textual datasets from ZenoHub [1], identifying 17 slicing criteria with their descriptions, examples, frequency, and creation methods—almost all are based on existing metadata or can be implemented with simple Python programs.**

With curated data, ML practitioners perform data analysis, model training or prompt engineering, and model evaluation (with potential debugging) in multiple iterations. Afterwards, the model can be deployed, with further model monitoring and updates based on production data. All these activities can benefit from *data slicing*.

Data slicing can help **model debugging**, as illustrated by our running example. After developers observe model mistakes, conduct error analysis, and formulate hypotheses, data slicing helps them validate the hypotheses on additional relevant examples [10, 56]. Once the hypotheses are validated, data slices can be used for further **model fixing**, via data pre-processing to construct useful features for model training, targeted data augmentation, or even as guidance for prompting [63].

Data slicing is also useful for fine-grained **model evaluation**, where multiple behavioral aspects are systematically examined [9, 44], as also recognized in the software engineering community [4]. Rather than error-chasing as in model debugging, developers have specific upfront behavioral aspects for model evaluation. The behavioral aspects can come from the developer's intuition, but can also be elicited from deliberate requirements analysis [e.g., 5, 59]. In contrast to traditional benchmarking, where practitioners only examine model accuracy on a single static benchmark, fine-grained model evaluation can expose nuanced model strengths and weaknesses and help pinpoint areas for model improvement. In our running example of toxicity detection, the developer might want to see if the model treats certain demographic groups systematically unfairly (e.g., regarding race, gender, religion) and use data slicing to identify many concrete examples for each subgroup.

This naturally extends to continuous **model monitoring**, where developers track model performance on multiple aspects through data slicing [41]. Developers can build regression test suites from data slices [32], as well as continuously analyze new production data. From model monitoring, developers can fix degrading aspects when needed, and potentially discover new data patterns (e.g., new hateful slang) for slicing when there is a distribution shift.

In earlier ML engineering phases, data slicing can contribute to **data curation**, as the insights and data produced from debugging, evaluation, and monitoring can be used to inform which slices are under-performing or under-represented and hence guide what data to curate [5].

For all these activities, the key motivation for data slicing is to generalize from individual data points to the underlying systematic problems. This generalization is necessary because model evaluation looks at accuracy in distributions rather than at mistakes for individual inputs – a single model mistake is not considered a bug [24]. This is in stark contrast to traditional software testing, where one single error is considered a bug that can be worth fixing. Parallels to data slicing, however, do exist in software testing: Equivalence class testing [49] divides the input space into several partitions and creates test cases for each partition, akin to how data slicing partitions datasets into slices; strong equivalence class testing explores interaction across these dimensions, which is also explored for data slicing [4, 39]. Despite the parallels, data slices are expected to come from distributions that practitioners care about, rather than from any failing inputs as in software testing. This key distinction from software testing is why ML engineering benefits from data-centric approaches like data slicing and dedicated innovations like SemSlicer.

*Status quo and limitations.* Despite being useful for many activities, (semantic) data slicing is not well-supported. Data slicing often assumes access to a set of existing features, but relevant features are not always available for users' slicing criteria, limiting developers in practice to create only "easy" slices. We found strong evidence for this by analyzing the 20 most popular projects on ZenoHub [1], a platform to share model evaluation results with built-in support for slicing, where we observed that developers almost exclusively create *programmatic* slicing functions (e.g., string length, question type) that are easy to implement, as shown in Table 1.

In contrast, various activities by practitioners and researchers suggest that developers often *want* to conduct *semantic* data slicing that cannot be matched with simple patterns: For example, Naik et al. [36] identified eight different error hypotheses for natural language inference models, with half of them being hard to cover with programmatic slicing (e.g., *"contradicting sentence pairs containing antonyms are hard to classify," "sentence pairs relying on real-world knowledge are hard to classify"*). Similarly, for machine

translation, Karpinska and Iyyer [25] report 15 mistranslation hypotheses annotated by humans, most of which are hard to cover with programmatic slicing (e.g., *"translations that change factuality," "translations that are overly literal"*). Ribeiro et al. [44] created a set of 41 functionality tests for three different NLP tasks, with 32 of them hard to implement using programmatic slicing (e.g., *"author sentiment is more important than that of others"*).

Indeed, precisely because of the difficulty of semantic data slicing, developers resort to synthetic data generation to obtain slices [e.g., 36, 44] or crowdsourcing to obtain extra annotations for slicing [e.g., 25]. To bridge this fundamental lack of support for semantic data slicing, SemSlicer employs LLMs as powerful tools to generate semantic slicing functions from any user criteria.

*From crowdsourcing to automated semantic slicing.* Recent advancements in LLMs have spurred interest in replacing crowdworkers with LLMs for various tasks [57]. The NLP community has explored using LLMs for data annotations [64], where researchers manually engineer their prompts for specific tasks. These annotations can be used as ground-truth labels for model training [20, 52], as noisy labels for weak supervision [48, 62], or as additional clues for model inference [34]. They have found LLMs often have similar, and sometimes even better performance than crowdworkers [11].

SemSlicer is motivated by this trend and aims to produce semantic slicing functions that traditionally can only be done by crowdworkers at a high cost. However, unlike the existing work that relies on manual prompt engineering for specific annotation tasks, SemSlicer is a unifying framework that can be applied to *any* slicing criteria, with a rich set of configuration options for constructing and optimizing the slicing prompt.

## 3 SEMANTIC DATA SLICING

SemSlicer is a framework for *semantic data slicing*, allowing users to create slicing prompts from specified slicing criteria, and use them to generate data slices. In this section, we will describe (1) the design dimensions of SemSlicer and (2) our system design and implementation details.

### 3.1 Design Dimensions

We design SemSlicer considering four dimensions: slicing accuracy needed (`accuracy`), latency expected (`latency`), human effort available (`human-effort`), and computational resources available (`compute`).

*3.1.1 Slicing accuracy needed.* Intuitively, we would want higher slicing `accuracy`, as it makes the observed slices more reliable. However, as we will demonstrate in Section 4.4, moderately accurate slices can also be useful for downstream use cases like model evaluation, as long as they can reliably detect where the model under-performs, making it possible to consider `accuracy` one of the four trade-off dimensions.

*3.1.2 Slicing latency expected.* Our second dimension, `latency`, considers how fast slicing should be. Depending on the downstream use cases, users can trade off `latency` for other dimensions, or vice versa. For example, interactive model debugging would expect a lower `latency`, while systematic model evaluation or monitoring can accept a higher `latency`.

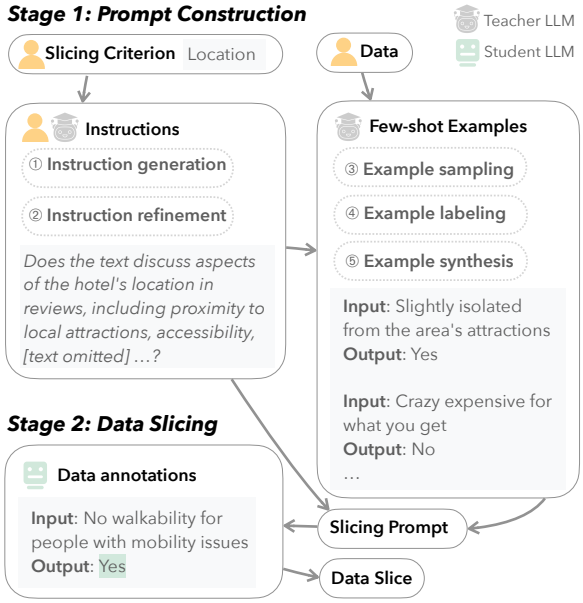


**Figure 3: SemSlicer's workflow: The user first specifies a slicing criterion (keywords, descriptions, etc.) and provides a dataset to slice. SemSlicer will ① construct and ② refine a classification instruction from the slicing criterion, optionally with human in the loop. SemSlicer will then ③ sample and ④ label few-shot examples, with ⑤ synthetic examples generated if needed. Finally, SemSlicer uses the produced prompt to annotate the dataset and create the slices.**

*3.1.3 Human effort available.* Our third dimension considers how much `human-effort` SemSlicer requires, which also depends on the use cases. For example, for interactive model debugging, we might want to prioritize lower `latency` over lower `human-effort`, as users can put more effort into shaping the slicing function (through prompting) but would expect faster interaction. In contrast, for model evaluation, we might prioritize `human-effort` over `compute` such that evaluation can scale to a larger number of slices.

*3.1.4 Computational resources available.* Our last dimension considers the practical concerns of how much `compute` is available for data slicing. Low computational cost is important for scaling up SemSlicer in practice. When resources are limited, users can accept lower `accuracy` to accommodate available resources, by using a smaller model or having a simpler prompt construction pipeline.

*Design trade-off.* There is no optimal design for all four dimensions: We often need to make trade-offs depending on the downstream use cases. Our system design aims to allow users to easily trade off along these dimensions, through using LLMs of different sizes and capabilities (`accuracy` vs. `compute`/`latency`), having a human in the prompt construction loop (`accuracy`/`compute` vs. `human-effort`), or having different setups of few-shot examples (`accuracy` vs. `compute`/`latency`). We will discuss these trade-offs in more detail in Section 3.2.

### 3.2 System Design

*System overview.* At a high level, SemSlicer runs in two stages (as depicted in Figure 3). In the first *prompt construction* stage,

```
A user is exploring a dataset to identify a subset that matches
      their goal.
Your job is to help the user craft a classification question. Answer
       ONLY with the question.

---

Follow the following format:

User goal: {{instruction}}
Suggestion: {{suggestion}}

---

[examples omitted]

User goal: {instruction}
```

**(a) Prompt for ① instruction generation.**

```
I have a classification instruction that does not perform well. Can
       you give me some suggestions to improve the instruction?

---

Follow the following format:

Instruction: {{instruction}}
Suggestion: {{suggestion}}
Revised instruction: {{instruction}}

---

Instruction: {instruction}
```

**(b) Prompt for ② instruction refinement.**

```
{question} Answer ONLY yes or no.

{examples}

Text: {text}
Answer:
```

**(c) Prompt for ④ example labeling and ⑥ slice labeling.**

```
{question} Answer ONLY yes or no.

{examples}

Write {n} examples with a '{label}' answer to the question above,
      following the format below.

Text: {{text}}
Answer: {{answer}}
```

**(d) Prompt for ⑤ example synthesis**

**Figure 4: Prompt templates used in SemSlicer.**

SemSlicer constructs a slicing prompt for a user-provided slicing criterion. SemSlicer will first ① generate and ② refine a classification instruction, and then ③ sample, ④ label, and optionally ⑤ synthesize few-shot examples. Depending on the configuration, some steps can be skipped or customized. In the second *data slicing* stage, SemSlicer will annotate the dataset using the constructed prompt and produce a corresponding slice.

*Slicing function components.* The final slicing function produced by SemSlicer will consist of three components: The model (and inference settings) used, the instructions specified, and the few-shot examples provided (see Figure 2, bottom). All three components can greatly impact slicing `accuracy` and `latency` and need different levels of `human-effort` and `compute`.

- **Model**: Model is the most critical component as LLM capabilities can greatly impact task `accuracy`. In SemSlicer, we leverage LLMs in many different steps, and for each step, we need to make trade-offs between `accuracy` and `compute/latency` to choose an appropriate model.
- **Instructions**: Instructions state what a model should do, often in the form of a classification question in SemSlicer (see example instruction in Figure 3). LLMs are shown to have strong instruction-following capabilities [38], but the quality of responses depends on the exact instructions provided [51]. SemSlicer supports using LLMs to help generate and refine instructions, with a human in the loop, inspired by recent advancements in automated prompt engineering [40, 66]. Turning this option on can improve `accuracy` at the cost of more `human-effort` or `compute`.
- **Few-shot examples**: Few-shot examples are demonstrations that show how a model should respond to a task input (see example input/output pairs in Figure 3). They also have a strong impact on LLM performance [8, 31]. SemSlicer supports a rich set of tools to construct few-shot examples, such as different sampling strategies, labeling strategies, and synthetic input generation. Users can make trade-offs between `accuracy` and `compute/latency` by choosing the right set of configurations.

Next, we detail our design choices, rationale, and implementations for these three components in SemSlicer.

*3.2.1 Model.* We design SemSlicer such that different steps and stages can flexibly leverage different LLMs, allowing trade-offs between `accuracy` and `compute/latency`. The general rationale behind our design is that a more powerful model can produce higher-quality outputs, but would increase the compute needed, incurring higher cost (and latency), while a less powerful model would be cheaper yet less accurate. Users should have the agency to decide which model fits best for their use case, according to their specific constraints.

In SemSlicer, LLMs are used in almost every step: In ① **instruction generation**, ② **instruction refinement**, and ⑤ **example synthesis**, we need a model with human-like creativity and capabilities to generate and refine instructions, as well as synthesize new examples. In ④ **example labeling**, we need a model with high-quality classification capabilities for small-scale labeling. In **data slicing**, we need a model with good-quality classification capabilities for larger-scale labeling.

*Implementation.* In our evaluation, we use `gpt-4-turbo-preview` for steps ①②⑤ (temperature=1) and step ④ (temperature=0),[3] as GPT-4 is known to be among the strongest LLMs available at the

[3]Temperature is a parameter commonly used to control LLM outputs. A higher temperature makes the model more "creative," while a lower temperature makes the model more predictable. We use temperature=1 for creativity as recommended [37], while temperature=0 is a common setup for classification tasks.

time of our implementation but also incurs higher cost and latency. These steps are all in the prompt construction stage and hence executed less frequently per slice.

We use `flan-t5-xxl` for the second stage (temperature=0), as the model has strong classification capabilities [12] and is of a moderate size with 11B parameters, hence lower cost and latency, which is crucial in the data slicing stage as the entire dataset is annotated. We further use 8-bit quantization to speed up model inference.

*3.2.2 Instructions.* We design SemSlicer to allow more LLM-based automated prompt engineering (`compute`), with a flexible human-AI collaboration mechanism (`human-effort`) to improve instruction quality (`accuracy`). This is achieved through two steps: ① **instruction generation**, to produce a good initial instruction, and ② **instruction refinement**, to further refine the produced instruction.

The first step aims to produce an initial instruction from a user-provided slicing criterion. Depending on what users provide, SemSlicer provides options for (1) template-based construction and (2) LLM-based generation:

(1) *Template*: When users provide a criterion in simple phrases, SemSlicer can use the generic template *"Is the text related to {concept}?"* to translate a criterion into an instruction. We observe this simple template works very well for many slicing tasks (cf. Section 4).
(2) *LLM-generated*: When users provide more detailed descriptions, SemSlicer can leverage an LLM, which offers strong information processing capabilities, to generate an initial instruction.

The produced instruction provides a good starting point, which can be further refined in the second step. This can be achieved through (1) human post-editing, (2) LLM-based refinement, or (3) a mixture of both. For LLM-based refinement, our design is inspired by Self-Refine [33], where we ask an LLM to provide suggestions and revisions on a to-be-revised instruction. This refinement step can increase `accuracy` at the cost of more `human-effort` or `compute`.

*Implementation.* We prompt an LLM for both ① **instruction generation** and ② **instruction refinement** (Figure 4).

When using the instruction in slice labeling, SemSlicer deliberately instructs the model to only produce a label token (Figure 4), without any intermediate reasoning steps (e.g., as in chain-of-thought prompting [54]). This design is based on our observation that the intermediate steps sometimes help `accuracy`, yet incur high `latency`, which scales linearly with the number of output tokens (`accuracy` vs. `latency`).

*3.2.3 Few-shot examples.* The design of SemSlicer automates the construction of few-shot examples, which demonstrate how a model should label a slice. Users can make trade-offs between `accuracy` and `compute`/`latency` by controlling different options, including (1) whether they need few-shot examples at all, (2) how many examples they need, (3) where the examples are from, and (4) how to label the examples. The more examples they use in the slicing prompt, the higher `compute` cost and `latency`.

As a first step to construct few-shot examples, SemSlicer needs to curate a set of example inputs. Assuming access to user-provided data, SemSlicer provides different ③ **sampling strategies**: (1) random sampling and (2) diversity sampling. We support diversity sampling as we observe that random sampling can miss examples from smaller slices, which biases the second stage of **data slicing**.

With the sampled example inputs, SemSlicer next labels the examples with the generated slicing instruction. In this step, we need to bootstrap from zero-shot labeling (i.e., without any examples), as there are usually no existing labels on a user's slicing criteria, which can be arbitrary. SemSlicer supports different ④ **labeling strategies**, (1) student-label and (2) teacher-label, to accommodate different trade-off decisions (`accuracy` vs. `compute`). A teacher model is a stronger LLM (e.g., GPT-4), whose labels are usually more accurate but also more expensive, while a student model is a smaller LLM that is less accurate but also cheaper. Using teacher-labeled examples can effectively distill a teacher model's knowledge about some particular slicing criterion to a student model.

In the last step, SemSlicer creates ⑤ **optional synthetic examples** to balance the in-slice and out-of-slice examples, as sometimes sampled examples can be extremely imbalanced for small slices, leaving no in-slice demonstrations for data slicing. To create synthetic examples that look similar to the real dataset, SemSlicer queries an LLM to write extra examples of the underrepresented label, *given sampled examples*. The given examples can condition the LLM on what inputs (style, content, etc.) it should generate.

*Implementation.* We implement diversity sampling by (1) vectorizing user-provided data using SentenceTransformer embeddings [42] and (2) clustering the data into $N$ clusters with KMeans and selecting one example from each cluster. This strategy produces semantically different clusters and hence diverse examples.

In our evaluation, we set the number of few-shot examples to 8, following existing work [12]. For ④ example labeling, we experiment with both a teacher model (`gpt-4-turbo-preview`) and a student model (`flan-t5-xxl`). For ⑤ example synthesis, we use `gpt-4-turbo-preview` (Figure 4).

*3.2.4 Data slicing.* With the slicing prompt constructed, the final data slicing step applies the prompt using a student model—we use `flan-t5-xxl` in our evaluation. This step produces slicing annotations for the entire dataset.

## 3.3 System Interface

SemSlicer is packaged as a Python library. As shown in Figure 2, a user first provides a dataset to slice (line 13) and a slicing criterion (line 14). Next, the user specifies what slicing function they want to generate with the desired configurations (line 16), with fine-grained control on each component of SemSlicer. SemSlicer will generate a slicing prompt, following the workflow summarized in Figure 3, and produce a slicing function (line 24), which can be further applied on any datasets (line 26). The entire process can be easily batched and extended to multiple slices.

The above example demonstrates how SemSlicer can support workflows with minimal `human-effort`. However, as we have discussed, humans can easily provide more guidance by (1) providing more detailed slicing instructions instead of simple keywords

| Method | Configurations | | | | | Cost/Slice |
|---|---|---|---|---|---|---|
| | Few-shot (fs) Examples | | | Instructions | | |
| | Input source | Input sampler | Output labeler | Source | Refinement | |
| $M_{\text{zero-shot}}$ | - (zero-shot, zs) | - | - | template | - | $0.26 |
| $M_{\text{few-shot}}$ | provided | random | student | template | - | $1.68 |
| $M_{\text{fs-div}}$ | provided | diversity (div) | student | template | - | $1.68 |
| $M_{\text{fs-teacher}}$ | provided | diversity | teacher | template | - | $1.69 |
| $M_{\text{fs-syn}}$ | provided + synthesized (syn) | diversity + balanced | teacher | template | - | $1.70 |
| $M_{\text{zs-model}}$ | - | - | - | model | - | $0.26 |
| $M_{\text{zs-tmodel}}$ | - | - | - | template (t) | model | $0.26 |
| $M_{\text{zs-hai}}$ | - | - | - | human + template | human + model (hai) | $8.13 |
| $M_{\text{fs-hai}}$ | provided | diversity | teacher | human + template | human + model (hai) | $9.56 |
| Crowdworker | - | - | - | human | - | $432.00 |

**Table 2: Experiment configurations: We selected 9 representative configurations of SemSlicer, following a fractional factorial design [3]. We estimate that SemSlicer costs from $0.26 to $9.56 to generate one slice from a dataset of 6k examples (CivilComments). In contrast, using crowdworkers for the same task would cost $432.00, which is 44x to 1661x more than SemSlicer (see the appendix for how we estimate cost).**

(line 14), (2) post-editing the generated prompt (line 23), or (3) iterating the entire process for a few rounds.

## 4 EVALUATION

First, to demonstrate feasibility and practicality, we evaluate our method's ability to produce accurate slices and its associated cost across *different* system configurations:

- **RQ1**: How accurate are SemSlicer's predicted slices across different configurations?
- **RQ2**: How much cost/latency does it take to produce the slices across different configurations?

Next, to demonstrate usefulness for downstream usage, we evaluate our method's ability to assist model evaluation:

- **RQ3**: How useful is SemSlicer for model evaluation?

### 4.1 Experiment Setup

*Datasets.* To evaluate our method, we collect existing datasets with *ground truth* slices, as there are no existing benchmarks on semantic data slicing. We look at three different strategies to collect suitable datasets:

- *Human annotations.* Sometimes dataset curators ask humans (usually crowdworkers) to annotate existing datasets with *additional* attributes of input texts (e.g., ethnicity groups referenced in texts). These attributes make plausible slices that are hard to obtain without human annotations. These slices are the most realistic (though potentially noisy due to crowdworker mistakes and subjectivity [2]), but they are challenging to find due to the high cost of human annotation.
  We use the existing CivilComments [7] dataset—the task is toxicity detection, but in addition to the toxicity label, the dataset has been annotated by crowdworkers with additional attributes regarding referenced demographic groups on five categories (gender, sexual orientation, religion, race, and disability) with 23 concrete attributes (e.g., Black, Christian). We derive slices from these additional attributes. To reduce experiment cost, we randomly sampled 6000 examples and used the 8 largest slices.
- *Synthetic data.* For model testing and evaluation [44], developers often create synthetic test data for different subgroups of the target population. We can treat data from each subgroup as a slice. These slices are accurate by construction, as they are created specifically for each subgroup, but can be less realistic due to their synthetic nature.
  We use two datasets created with this strategy, HateCheck [45] and AdaTest [43]. The first is a hate speech test suite (n=3728) with comments for 7 different subgroups (e.g., women, trans). The second is a sentiment analysis test suite (n=196) with data for 6 aspects (e.g., price, location) of a hotel review.
- *Metadata.* Many datasets come with metadata produced as part of data collection, such as tags on QA websites from which data was scraped. We can create slices from such metadata, but these slices may be less accurate as different categories (e.g., clothes vs. sports) might overlap semantically, leaving some examples missing from the slices.
  We use the existing Amazon [6] sentiment analysis dataset, which in addition to the sentiment label, contains metadata about what product category was reviewed (e.g., books, electronics). To reduce experiment cost, we randomly sampled 6000 examples and used the 5 largest slices.

To summarize, we selected four datasets with 26 ground-truth slices—see Table 3 for their names and proportions. These datasets cover different uses of slicing: slicing on sensitive attributes for fairness (HateCheck and CivilComments) and slicing on topics/domains for fine-grained model evaluation (AdaTest and Amazon). We collected all slicing criteria (the inputs to SemSlicer) directly from the datasets, or from the descriptions in the associated papers.

*Configurations.* To understand how different configuration options impact the accuracy and cost of SemSlicer, we selected and analyzed 9 representative configurations of SemSlicer, following a fractional factorial design [3] to cover all values in each dimension (as shown in Table 2). These configurations cover whether to use few-shot examples (zero-shot vs. few-shot), how few-shot inputs are collected (provided vs. synthetic), sampled (random vs. diversity), and labeled (student vs. teacher), as well as how instructions

| Dataset | Slice | Frac. (%) | Slicing F1 score (%) | | | | | | | | | Task perf. (%) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $M_{zero\text{-}shot}$ | $M_{few\text{-}shot}$ | $M_{fs\text{-}div}$ | $M_{fs\text{-}teacher}$ | $M_{fs\text{-}syn}$ | $M_{zs\text{-}model}$ | $M_{zs\text{-}tmodel}$ | $M_{zs\text{-}hai}$ | $M_{fs\text{-}hai}$ | gold | pred. |
| **HateCheck** | women | 13.7 | 90.7 | 92.7 | 92.6 | 92.8 | 91.7 | 92.3 | 89.5 | 90.5 | 94.6 | 79.4 | 82.6 |
| | trans | 12.4 | 83.6 | 79.5 | 78.3 | 95.1 | 94.7 | 95.1 | 75.7 | 90.3 | 96.5 | 53.3** | 52.7** |
| | gay | 14.8 | 68.3 | 69.0 | 69.5 | 69.7 | 68.8 | 85.0 | 69.2 | 88.4 | 71.7 | 70.6 | 63.8** |
| | black | 12.9 | 95.2 | 95.4 | 95.1 | 95.1 | 95.7 | 95.2 | 86.3 | 94.9 | 95.1 | 70.7 | 71.6 |
| | disabled | 13.0 | 68.1 | 77.9 | 67.3 | 67.4 | 81.6 | 71.4 | 69.2 | 89.4 | 73.3 | 47.3** | 48.7** |
| | muslim | 13.0 | 97.7 | 97.1 | 95.8 | 95.4 | 93.9 | 97.1 | 95.8 | 97.2 | 95.4 | 78.3 | 79.9 |
| | immigrant | 12.4 | 80.0 | 77.2 | 66.8 | 82.2 | 68.5 | 94.5 | 21.9 | 94.9 | 93.5 | 71.5 | 74.3 |
| | avg | - | 83.4 | 84.1 | 80.8 | 85.4 | 85.0 | 90.1 | 72.5 | 92.2 | 88.6 | 69.4 | 69.4 |
| **AdaTest** | room | 24.9 | 59.3 | 84.9 | 88.2 | 89.6 | 87.4 | 15.4 | 70.9 | 84.7 | 88.6 | 66.0 | 65.3 |
| | location | 16.9 | 49.1 | 74.1 | 73.6 | 82.7 | 80.5 | 28.6 | 48.5 | 82.4 | 87.9 | 62.5 | 58.1 |
| | price | 18.5 | 95.9 | 95.8 | 95.8 | 94.3 | 94.3 | 95.8 | 24.4 | 97.2 | 95.8 | 65.7 | 62.9 |
| | restaurant | 17.5 | 57.1 | 76.5 | 65.1 | 78.1 | 62.4 | 59.2 | 58.3 | 78.6 | 93.8 | 63.6 | 74.2 |
| | service | 15.9 | 37.3 | 49.2 | 49.6 | 54.1 | 48.8 | 45.8 | 43.3 | 59.4 | 55.0 | 80.0 | 76.5 |
| | pool | 6.3 | 85.7 | 90.9 | 90.9 | 73.7 | 90.9 | 80.0 | 73.7 | 90.9 | 90.9 | 50.0 | 57.1 |
| | avg | - | 64.1 | 78.6 | 77.2 | 78.7 | 77.4 | 54.1 | 53.2 | 82.2 | 85.3 | 66.1 | 66.1 |
| **Amazon** | book | 61.4 | 95.9 | 96.5 | 96.3 | 97.0 | 96.7 | 95.9 | 96.6 | 94.7 | 95.1 | 68.7 | 68.7 |
| | movie | 7.0 | 53.1 | 55.7 | 55.9 | 59.1 | 56.3 | 58.1 | 55.0 | 56.0 | 81.1 | 69.6 | 66.5 |
| | home&kitchen | 6.3 | 52.1 | 47.2 | 48.5 | 48.3 | 45.8 | 50.3 | 59.9 | 59.7 | 48.1 | 73.3 | 69.1 |
| | electronics | 5.8 | 52.1 | 54.3 | 57.8 | 60.0 | 56.6 | 52.1 | 62.8 | 61.7 | 64.9 | 74.7 | 72.4 |
| | clothing | 5.4 | 63.8 | 72.0 | 65.9 | 72.8 | 66.3 | 66.1 | 75.1 | 74.8 | 79.1 | 71.8 | 72.5 |
| | avg | - | 63.4 | 65.2 | 64.9 | 67.4 | 64.3 | 64.5 | 69.9 | 69.4 | 73.7 | 69.8 | 69.8 |
| **CivilComments** | male | 2.5 | 12.3 | 30.6 | 32.3 | 29.8 | 5.1 | 10.6 | 25.3 | 9.3 | 10.9 | 89.3 | 90.7 |
| | female | 2.9 | 30.1 | 40.9 | 40.9 | 42.5 | 17.3 | 30.9 | 33.2 | 31.5 | 36.3 | 87.3** | 89.4* |
| | homosexual | 0.5 | 31.3 | 38.8 | 43.5 | 40.9 | 40.6 | 35.2 | 25.3 | 41.8 | 50.6 | 82.8* | 87.8* |
| | christian | 2.3 | 53.1 | 59.2 | 56.1 | 51.0 | 55.3 | 54.3 | 57.0 | 51.3 | 51.5 | 91.9 | 93.8 |
| | jewish | 0.5 | 50.0 | 55.0 | 69.7 | 69.1 | 62.3 | 50.0 | 48.4 | 49.5 | 85.7 | 92.6 | 96.4 |
| | muslim | 1.2 | 60.9 | 71.3 | 69.3 | 73.1 | 79.2 | 70.7 | 71.1 | 78.2 | 82.0 | 87.3* | 88.5* |
| | black | 0.8 | 43.2 | 54.3 | 64.5 | 69.1 | 66.1 | 31.0 | 44.4 | 44.7 | 71.0 | 75.0** | 79.0** |
| | white | 1.4 | 73.0 | 81.5 | 83.6 | 78.7 | 85.5 | 67.5 | 32.5 | 78.1 | 85.2 | 81.4** | 78.1** |
| | avg | - | 44.2 | 54.0 | 57.5 | 56.8 | 51.4 | 43.8 | 42.1 | 48.1 | 59.2 | 93.5 | 93.5 |
| - | avg | - | 63.0 | 69.9 | 69.7 | 71.6 | 68.9 | 62.6 | 58.2 | 71.9 | 75.9 | - | - |

best good poor

$^{**}p < 0.01$, $^{*}p < 0.05$

**Table 3: We collected 4 datasets and 26 slices for our evaluation. These slices represent a 0.5% to 61.4% fraction of the entire dataset. We found that SemSlicer achieves an average F1 score of up to 75.9% (achieved by $M_{fs\text{-}hai}$), with significant improvement from few-shot examples (+4.0% vs. $M_{zs\text{-}hai}$) and human interventions (+4.3% vs. $M_{fs\text{-}teacher}$) (RQ1). We also found that SemSlicer can recover 7 out of 7 slices that have significantly lower downstream task performance, with only one false positive (RQ3).**

are generated (template vs. model vs. human+template) and refined (model vs. human+model).

For the configurations with human interventions ($M_{zs\text{-}hai}$ and $M_{fs\text{-}hai}$), we have one of the authors (1) write down slicing descriptions for the model and (2) post-edit the model-refined instructions.

Note that here we deliberately compare different configurations of SemSlicer, instead of comparing it to a baseline using regular expression. This is because programmatic slicing performs too poorly to make a meaningful baseline: For example, our analysis shows that direct keyword matching on the literal keyword "location" yields low recall (0.059), and a refined regex approach (r"location|walk|far from|close to|neighborhood|near") only improves recall to 0.65. Even such a mediocre performance already requires extensive user labor—we crafted the regex after carefully inspecting 30% of examples from the ground-truth "location" slice, which results in overfitting and demonstrates the limitations of rule-based systems. Including such an obviously poor baseline would not have added meaningful value to the evaluation.

*Threats to Validity.* Despite best efforts, the datasets we used are not perfect, with noisy labels or less realistic inputs as explained above. The human-in-the-loop configurations are also limited by the authors' prompt writing expertise. As is common in these kinds of studies in real-world settings, our human-subject case study trades off lower internal validity for higher external validity, in a context with limited control over the setting and limited ability to perform repeated independent observations. Generalizations beyond our evaluation results should be done with care.

SemSlicer uses LLMs at multiple steps. While LLMs can be non-deterministic and make the results less reliable, we deliberately used temperature=0 setting for the data slicing step. That is, repeatedly sending the same prompt to the same LLM will produce the same answer. We also used an open-source LLM, making this step fully reproducible. For instructions and synthetic examples in prompt construction, the variances from LLM non-determinism are smoothed over as we average results from 26 slices. [4]

## 4.2 RQ1: Slicing Accuracy

*Setup.* We measure *slicing accuracy* for each slice, that is, how well the slices generated by SemSlicer correspond to the ground-truth slices in the dataset. We measure slicing accuracy using F1-score, an established metric for imbalanced datasets, as slices often represent a small fraction of the entire dataset (ranging from 0.5%

[4]To better understand the evaluation variance for individual slices, we ran $M_{zs\text{-}model}$ five times on seven HateCheck slices. We observed that the standard deviation ranges from 0.0017 to 0.061, with maximal F1 differences up to 14%. After averaging, the standard deviation is 0.013, and the maximal F1 difference is only 3.5%, supporting our point that the averaged results reduce evaluation variances.

| Dataset | Annotations/sec. | | Tokens/annotation | |
|---|---|---|---|---|
| | zero-shot | few-shot | zero-shot | few-shot |
| **HateCheck** | 12.25 | 10.99 | 32.12 | 178.73 |
| **AdaTest** | 14.56 | 10.84 | 27.45 | 111.79 |
| **Amazon** | 14.79 | 6.01 | 131.40 | 648.90 |
| **CivilComments** | 14.46 | 6.45 | 93.17 | 626.54 |
| **Average** | 14.07 | 7.40 | 92.47 | 523.79 |

**Table 4: SemSlicer can produce slices at a fast speed: Depending on the exact setup (zero-shot vs. few-shot) and dataset characteristics (tokens per annotation), SemSlicer can annotate from 6 to 14 examples per second. As an example, annotating the CivilComments dataset (n=6k) takes around 15.5 minutes, using 2 A6000 GPUs.**

to 61.4% in our evaluation, see Table 3). We additionally compute the average F1-score across all slices from each dataset, as well as average F1-score overall.

*Result: SemSlicer produces accurate annotations with 75.9% average F1-score across all slices in the best configuration.* SemSlicer produced the most accurate slices with 75.6% average F1-score in configuration $M_{\text{fs-hai}}$ with few-shot examples and human+model refinement, as shown in Table 3. For the configurations without human intervention, we found $M_{\text{fs-teacher}}$ has the highest average F1-score of 71.6%. As discussed earlier, SemSlicer does not need perfect accuracy to generate useful slices—our results for RQ3 will show that the current level of accuracy can already reliably detect under-performing slices.

*Result: Few-shot prompting improves accuracy by 7% on average.* Comparing few-shot configurations with zero-shot, we found few-shot prompting improves performance by 7% on average (Table 3). Among different few-shot setups, we found teacher labeling improves performance by 1.9% ($M_{\text{fs-teacher}}$ vs. $M_{\text{fs-div}}$), but observed a limited impact from sampling methods ($M_{\text{few-shot}}$ vs. $M_{\text{fs-div}}$) and a negative impact from synthetic examples ($M_{\text{fs-teacher}}$ vs. $M_{\text{fs-syn}}$). We hypothesize that this is because the generated synthetic examples are still too unrealistic to help later slice labeling.

*Result: Human intervention significantly improves accuracy.* We observe that LLM-based instruction generation and refinement have an unstable impact compared to simple templates: They sometimes generate good instructions that improve accuracy a lot (e.g., $M_{\text{zs-model}}$ for HateCheck) but can also generate bad instructions that hurt accuracy (e.g., $M_{\text{zs-tmodel}}$ for HateCheck). However, with human interventions, generated instructions significantly improve accuracy, with an 8.9% increase for zero-shot ($M_{\text{zero-shot}}$ vs. $M_{\text{zs-hai}}$), and a 4.3% increase for few-shot ($M_{\text{fs-teacher}}$ vs. $M_{\text{fs-hai}}$). This shows that human feedback is particularly useful to guide instruction optimization in the right direction.

*Discussion.* Because we noticed that SemSlicer performs poorly on some slices of the CivilComments dataset, especially due to low precision, we investigated the reasons for this behavior. We sampled 40 false positive examples from four slices with low precision scores (*male*, *female*, *homosexual*, *christian*) under the $M_{\text{fs-teacher}}$ setting and manually inspected the examples: Surprisingly, we found that 24 out of 40 examples actually cover nuanced patterns that (we argue) should be in the slices, yet were missed by human annotators. These patterns are often related to the slicing criterion in a less direct way (e.g., "Pride" for *homosexual*; "scripture", "church" for *christian*). This shows that SemSlicer is not only comparable to human annotations but sometimes can even surpass them, echoing the findings from some existing work that LLM annotations can have higher quality than human annotations [20].

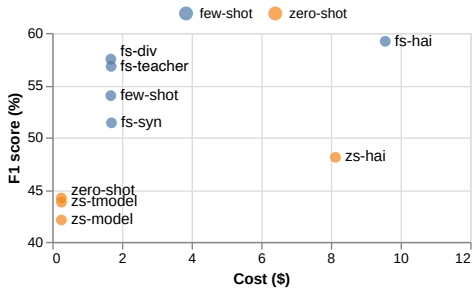


**Figure 5: We visualize F1 score and cost of each configuration (for CivilComments, n=6k), which shows a clear trend of two trade-offs: whether to use few-shot examples (higher `accuracy` with higher `compute`), and whether to have human in the loop (higher `accuracy` with more `human-effort`).**

## 4.3 RQ2: Cost and Latency

*Setup.* In our experiments, we use a local machine with 2 A6000 GPUs for running `flan-t5-xxl` and use OpenAI's APIs to query `gpt-4-turbo-preview`. We collected the end-to-end slicing latency and calculated the number of annotations per second as the normalized latency. To measure cost, we collected the number of input/output tokens for each LLM query and estimated the cost (in USD) based on charges from LLM providers. We estimated the human cost based on the average pay for crowdworkers and data scientists. The calculation details can be found in the appendix.[5]

*Result: SemSlicer can slice a dataset with 6,000 rows in 13.5 minutes with a cost of $1.70 per slice in the most accurate automated configuration.* We found SemSlicer can annotate 6 to 14 examples per second (Table 4). This translates into using 7.1 minutes (zero-shot) to 13.5 minutes (few-shot) for annotating one slice in a dataset of size of 6k (CivilComments). We estimate that the associated cost is about $0.26 (zero-shot) to $1.70 (few-shot) per slice for automated configurations, and up to $9.56 ($M_{\text{fs-hai}}$) for human-in-the-loop configurations (Table 2). In contrast, using crowdworkers for the same task would cost $432.00, which is 44x to 1661x more than SemSlicer but not necessarily more accurate.

*Result: SemSlicer allows flexible trade-offs between design dimensions through different configurations.* Comparing different configurations, we observed there are clear trade-offs between design dimensions among different configurations (Figure 5). Using few-shot examples can increase the cost by around $1.40 per slice and takes 91% more time (Table 4), with significant accuracy improvement by 7% on average (`accuracy` vs. `compute`/`latency`). Human interventions can increase the cost a lot ($7.87 per slice), but also come with significant accuracy improvement by 4.3% to 8.9% (`accuracy` vs. `human-effort`).

[5] https://github.com/malusamayo/SemSlicer/blob/main/appendix.md

*Discussion.* As shown in Figure 5 , there are trade-offs to make between different design dimensions. In production, the best configuration will depend on the dataset and the slicing criterion difficulty. For easier criteria, a simple zero-shot configuration would already work pretty well (e.g., the "price" slice in Table 3 has a 95.9% F1-score with $M_{\text{zero-shot}}$). For more difficult criteria, we recommend users to select configurations based on the following two principles:

- If there is more `compute` budget, always try few-shot examples. Use a teacher model to generate example labels if available.
- If there is more `human-effort` budget, allocate it to instruction refinement.

## 4.4 RQ3: Usefulness

We approach usefulness from two perspectives, by understanding (1) whether SemSlicer can help identify *known* under-performing slices and (2) whether SemSlicer can help practitioners conduct model evaluation and generate *additional* insights.

For the first part, we demonstrate usefulness if SemSlicer can reliably detect under-performing slices, a common use of data slicing (cf. Section 2). For the second part, we conduct a human-subject case study to triangulate the automated experiments, by showing that practitioners found SemSlicer useful in their real-world cases.

### 4.4.1 Can SemSlicer help identify under-performing slices?

*Setup.* A common use of slicing is to identify parts of the input space where the model under-performs. That is, we want to identify the slices in which their downstream *task performance* (i.e., the accuracy of the original task, such as toxicity detection) is statistically significantly worse than the average, even if the *slicing accuracy* is not perfect. Specifically, we want to evaluate whether our slicing accuracy is good enough to detect those same slices as under-performing that we would have detected with perfect slicing accuracy from the ground-truth attributes in our datasets.

To understand whether SemSlicer can identify under-performing slices, we measure *task performance* for each dataset or slice. For AdaTest, we reuse the reported predictions from their commercial sentiment analysis model and compare them against the labels in the dataset. For HateCheck, Amazon, CivilComments, we use popular models (listed in the appendix) from Hugging Face and compare their predictions against the labels in the dataset. For all datasets, we report *task performance* with the standard accuracy metric.

For our analysis, we determine whether a slice (ground truth or computed with SemSlicer) under-performs by comparing the slice's task performance against the overall task performance on the entire dataset, using Fisher's exact test to test whether the difference is statistically significant (p-value $\leq 0.05$). We then compare under-performing slices detected through SemSlicer with those detected on ground-truth slices. For this experiment, we use the most accurate automated configuration, $M_{\text{fs-teacher}}$.

*Result: SemSlicer identifies all 7 under-performing slices.* We found that 7 out of 26 slices are statistically significantly under-performing and SemSlicer is able to identify all of them (Table 3). In addition, SemSlicer identifies only one slice as under-performing that is not under-performing according to the ground-truth slice (slice *"gay"* in HateCheck; false positive). This demonstrates that SemSlicer's slicing accuracy is sufficient for the practical application of identifying under-performing slices.

### 4.4.2 Can SemSlicer help practitioners conduct model evaluation?

*Setup.* We approached researchers in our contact network to identify ML-related projects and selected one participant. The participant worked on a project to understand how LLM annotations align differently with different demographic groups. More specifically, the participant had annotators annotate the Social Acceptability [16] dataset and computed the correlations between model and human annotations on various slices of *annotator attributes*. The participant observed that LLMs (e.g., GPT-4) aligned better with Western, White, college-educated, and younger populations and was interested in using SemSlicer to tie correlations back to specific slices on *input text*, which they could not have done without support from semantic data slicing.

Before the study, we first discussed with the participant about what kinds of input text slices they would be interested to see for their dataset. The participant mentioned that slices concerning different demographics in the input text would be interesting—following this, we collected 6 slicing criteria for our case study (*"gender", "age", "nationality", "education", "religion", "ethnicity"*). We then used SemSlicer to generate the slices, using the most accurate automated setup of $M_{\text{fs-teacher}}$. Finally, we invited the participant for a one-hour study session, where the participant inspected the generated slices (visualized in Zeno [9]) and their model-human correlation scores in think-aloud mode. We collected their verbalized thoughts and feedback on the slices generated by SemSlicer in the process.

*Result: SemSlicer supports new opportunities in model evaluation and helps generate additional insights.* Looking at the slices, the participant quickly realized there was a gap between what they expected and what the slices showed. When inspecting the slice on gender, the participant found many examples are only superficially related to gender by mentioning gendered pronouns (which is the expected behavior of SemSlicer) but what they really cared about are the examples showing gender-related power imbalance. With this realization, the participant suggested that we can instead slice on *"power imbalance"* as slicing criteria, and its gender-related or age-related subsets. This anecdote reflects on a larger theme of how users often need to iterate on their slicing criteria, after realizations of their hidden requirements.

We reran SemSlicer to produce three slices (took 7 minutes end-to-end) and reached back to the participant. The participant found that, this time, they can agree with most of the examples in the slices. In a few cases, SemSlicer even nudged the participant to understand the nuances: *"[now] I could see it, especially the mother-in-law thing."* Further, the participant was able to generate *additional* insights that they could not have done in their original analysis. For example, the participant found that for text related to age-related power imbalance, the model aligns best with millennials, with a stark correlation drop for age groups above 40, which they did not observe in the overall dataset. Overall, the participant found SemSlicer useful to help them develop a more fine-grained understanding of their original results.

# 5 RELATED WORK

In Section 2, we already discussed the most closely related work on data slicing. Here, we *additionally* discuss related work on *automated prompt engineering* relevant to how we implemented SemSlicer, as well as how *slice discovery* can augment semantic slicing.

## 5.1 Automated Prompt Engineering

*Instructions.* Automated improvement of prompt instructions is an emerging direction that has attracted lots of attention. Zhou et al. [66] generate and paraphrase instruction candidates using LLMs and select instructions based on scores on labeled evaluation data. Pryzant et al. [40] improve instructions iteratively with erroneous examples. Both studies assume access to a labeled dataset. In contrast, SemSlicer's instruction generation and refinement are designed for a zero-label setting.

*Few-shot examples.* Few-shot examples are known to have a large impact on prompt performance [65], and few-shot example selection has been an active field of research. Liu et al. [30] propose to retrieve examples that are semantically *similar* to a test sample. Other work found the retrieved examples are often redundant and proposed to find *diverse* examples with high coverage [19, 61].

Most of the existing work assumes access to a labeled training dataset, while SemSlicer requires few-shot example selection without labels. Even though we cannot use these methods directly, they still inspired our design of the input sampler, where we provide the option to retrieve diverse inputs.

Closest to our work is universal self-adaptive prompting [53], where they select high-confidence examples (based on the self-predictions) as few-shot examples. Our early experimentation did not observe consistent improvement from this method and hence we did not report it in our evaluation. However, it is still available as an option for input sampling.

DSPy [26] is a framework that can automatically generate and optimize prompts for generic LLM pipelines. SemSlicer is designed specifically for data slicing as a standalone framework, but can potentially be implemented using existing frameworks like DSPy.

*Prompt selection.* Another line of work on zero-label prompt selection [60] aims to select a good prompt from multiple candidates. Existing work has explored selection using pseudo labels from prompt ensembles [29], perplexity scores [18], and mutual information [50]. Our early experimentation suggested limited improvement from these methods, and hence we did not use them in the final design.

## 5.2 Automated Slice Discovery

Automated slice discovery [13, 14] is an error analysis approach based on the idea of automatically identifying under-performing areas in the input space, often using clustering or dimensionality reduction techniques. These areas can be interpreted as slices and this line of work explores ways to identify meaningful names for those areas, which could then be considered as slicing criteria. However, these methods are designed to discover *under-performing* areas, without considering whether these are *complete* according to some human-interpretable abstraction. As pointed out by Johnson et al. [23], their produced slices can be misleading, in that models that under-perform on the found slice often do not under-perform on what would be a complete slice for that concept. In contrast, SemSlicer aims to identify *all* relevant examples for a given slicing criteria. Overall automated slice discovery is complementary to our approach as an *error analysis* technique to generate hypotheses (cf. Fig. 1), whereas we focus on creating slices to *validate hypotheses*.

# 6 CONCLUSION

In this work, we present SemSlicer, a framework that supports semantic data slicing. We provide a comprehensive view of how data slicing is broadly applicable in ML engineering and demonstrate how SemSlicer provides new opportunities missed by existing programmatic slicing methods. Our evaluation found SemSlicer can produce accurate slices at a low cost, with flexible trade-offs among different design dimensions, and is useful for model evaluation.

Insights from our work provide new research opportunities:

*Semantic slicing at a reduced cost.* In its current design, SemSlicer costs $0.26 to $1.70 to automatically produce slices from a dataset of 553k tokens. This is much cheaper than recruiting crowdworkers but can still incur substantial cost for bigger datasets, longer inputs, or more slices. One promising direction is to explore retrieval-based methods [27] to filter out irrelevant data points and reduce the annotations needed. Another is to train small customized models specifically for the task of data slicing.

*Semantic slicing with higher accuracy.* SemSlicer has implemented a wide range of configuration options. However, the field of automated prompt engineering is evolving quickly. Future work can incorporate additional emerging research findings and test their combinations. For example, one direction is to build an agentic framework [58] for semantic slicing, where the slicing agent actively refines instructions and few-show examples, based on automated or LLM-generated feedback.

*Interactive semantic slicing.* The current design of SemSlicer allows human intervention through editing slicing conditions and post-editing instructions. Future work can integrate SemSlicer in an interactive interface (e.g., Zeno [9]), and explore different interaction designs. For example, humans can annotate few-shot examples in an active learning loop [47], or provide verbalized feedback for instruction refinement. This way, humans have more agency in prompt construction, potentially leading to faster iterations.

# ACKNOWLEDGMENTS

We thank Claire Le Goues, Graham Neubig, Alex Cabrera, Xinran Zhao, Jenny Liang, Vijay Viswanathan, Manisha Mukherjee, and others for their feedback on this work. Kaestner and Yang's work was, in part, supported by NSF 2106853, 2131477, and 2206859. Yining's work was supported by the Top Open Program at Tsinghua University as an REU student at Carnegie Mellon. Lewis' work was funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center (DM24-1142). The work was also supported by gift funds from Amazon, Google Research, and Adobe Research.

# REFERENCES


[1] [n. d.]. Zeno AI Evaluation Platform. https://hub.zenoml.com/home

[2] Lora Aroyo and Chris Welty. 2015. Truth Is a Lie: Crowd Truth and the Seven Myths of Human Annotation. *AI Mag.* 36 (2015), 15–24. https://api.semanticscholar.org/CorpusID:6134326

[3] Rosemary A Bailey. 2008. *Design of comparative experiments*. Vol. 25. Cambridge University Press.

[4] Guy Barash, Eitan Farchi, Ilan Jayaraman, Orna Raz, Rachel Tzoref-Brill, and Marcel Zalmanovici. 2019. Bridging the gap between ML solutions and their business requirements using feature interactions. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 1048–1058. https://doi.org/10.1145/3338906.3340442

[5] Hamed Barzamini, Mona Rahimi, Murteza Shahzad, and Hamed Alhoori. 2022. Improving Generalizability of ML-Enabled Software through Domain Specification. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI* (Pittsburgh, Pennsylvania) *(CAIN '22)*. Association for Computing Machinery, New York, NY, USA, 181–192.

[6] John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Annie Zaenen and Antal van den Bosch (Eds.). Association for Computational Linguistics, Prague, Czech Republic, 440–447. https://aclanthology.org/P07-1056

[7] Daniel Borkan, Lucas Dixon, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. 2019. Nuanced Metrics for Measuring Unintended Bias with Real Data for Text Classification. In *Companion Proceedings of The 2019 World Wide Web Conference* (San Francisco, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 491–500. https://doi.org/10.1145/3308560.3317593

[8] Tom Brown, Benjamin Mann, et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[9] Ángel Alexander Cabrera, Erica Fu, Donald Bertucci, Kenneth Holstein, Ameet Talwalkar, Jason I. Hong, and Adam Perer. 2023. Zeno: An Interactive Framework for Behavioral Evaluation of Machine Learning. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 419, 14 pages. https://doi.org/10.1145/3544548.3581268

[10] Ángel Alexander Cabrera, Marco Tulio Ribeiro, Bongshin Lee, Robert Deline, Adam Perer, and Steven M. Drucker. 2023. What Did My AI Learn? How Data Scientists Make Sense of Model Behavior. *ACM Trans. Comput.-Hum. Interact.* 30, 1, Article 1 (mar 2023), 27 pages. https://doi.org/10.1145/3542921

[11] Jan Cegin, Jakub Simko, and Peter Brusilovsky. 2023. ChatGPT to Replace Crowdsourcing of Paraphrases for Intent Classification: Higher Diversity and Comparable Model Robustness. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 1889–1905. https://doi.org/10.18653/v1/2023.emnlp-main.117

[12] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. arXiv:2210.11416 [cs.LG]

[13] Greg d'Eon, Jason d'Eon, J. R. Wright, and Kevin Leyton-Brown. 2021. The Spotlight: A General Method for Discovering Systematic Errors in Deep Learning Models. *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency* (2021). https://api.semanticscholar.org/CorpusID:235727396

[14] Sabri Eyuboglu, Maya Varma, Khaled Saab, Jean-Benoit Delbrouck, Christopher Lee-Messer, Jared Dunnmon, James Zou, and Christopher Ré. 2022. Domino: Discovering systematic errors with cross-modal embeddings. *arXiv preprint arXiv:2203.14960* (2022).

[15] Michael Feffer, Anusha Sinha, Zachary C Lipton, and Hoda Heidari. 2024. Red-Teaming for Generative AI: Silver Bullet or Security Theater? *arXiv preprint arXiv:2401.15897* (2024).

[16] Maxwell Forbes, Jena D. Hwang, Vered Shwartz, Maarten Sap, and Yejin Choi. 2020. Social Chemistry 101: Learning to Reason about Social and Moral Norms. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 653–670. https://doi.org/10.18653/v1/2020.emnlp-main.48

[17] Gartner. 2023. Gartner Identifies the Top Strategic Technology Trends for 2024. (2023).

[18] Hila Gonen, Srini Iyer, Terra Blevins, Noah Smith, and Luke Zettlemoyer. 2023. Demystifying Prompts in Language Models via Perplexity Estimation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 10136–10148. https://doi.org/10.18653/v1/2023.findings-emnlp.679

[19] Shivanshu Gupta, Matt Gardner, and Sameer Singh. 2023. Coverage-based Example Selection for In-Context Learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 13924–13950. https://doi.org/10.18653/v1/2023.findings-emnlp.930

[20] Xingwei He, Zheng-Wen Lin, Yeyun Gong, Alex Jin, Hang Zhang, Chen Lin, Jian Jiao, Siu Ming Yiu, Nan Duan, and Weizhu Chen. 2023. AnnoLLM: Making Large Language Models to Be Better Crowdsourced Annotators. *ArXiv* abs/2303.16854 (2023). https://api.semanticscholar.org/CorpusID:257805087

[21] Zexue He, Marco Tulio Ribeiro, and Fereshte Khani. 2023. Targeted Data Generation: Finding and Fixing Model Weaknesses. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 8506–8520. https://doi.org/10.18653/v1/2023.acl-long.474

[22] Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving google's perspective api built for detecting toxic comments. *arXiv preprint arXiv:1702.08138* (2017).

[23] Nari Johnson, Ángel Alexander Cabrera, Gregory Plumb, and Ameet Talwalkar. 2023. Where Does My Model Underperform? A Human Evaluation of Slice Discovery Algorithms. *ArXiv* abs/2306.08167 (2023). https://api.semanticscholar.org/CorpusID:259165223

[24] Christian Kaestner. 2020. Machine Learning is Requirements Engineering — On the Role of Bugs, Verification, and Validation in Machine Learning. Blog.

[25] Marzena Karpinska and Mohit Iyyer. 2023. Large Language Models Effectively Leverage Document-level Context for Literary Translation, but Critical Errors Persist. In *Proceedings of the Eighth Conference on Machine Translation*. Association for Computational Linguistics, Singapore, 419–451. https://doi.org/10.18653/v1/2023.wmt-1.41

[26] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714* (2023).

[27] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 39–48.

[28] Pang Wei Koh et al. 2021. WILDS: A Benchmark of in-the-Wild Distribution Shifts. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 5637–5664.

[29] Chonghua Liao, Yanan Zheng, and Zhilin Yang. 2022. Zero-Label Prompt Selection. arXiv:2211.04668 [cs.CL]

[30] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What Makes Good In-Context Examples for GPT-3?. In *Workshop on Knowledge Extraction and Integration for Deep Learning Architectures; Deep Learning Inside Out*. https://api.semanticscholar.org/CorpusID:231632658

[31] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 8086–8098.

[32] Wanqin Ma, Chenyang Yang, and Christian Kästner. 2024. (Why) Is My Prompt Getting Worse? Rethinking Regression Testing for Evolving LLM APIs. In *Proceedings of the International Conference on AI Engineering - Software Engineering for AI (CAIN)* (Lisbon).

[33] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2024).

[34] Sachit Menon and Carl Vondrick. 2022. Visual Classification via Description from Large Language Models. In *The Eleventh International Conference on Learning Representations*.

[35] Danaë Metaxa, Joon Sung Park, Ronald E. Robertson, Karrie Karahalios, Christo Wilson, Jeff Hancock, and Christian Sandvig. 2021. Auditing Algorithms: Understanding Algorithmic Systems from the Outside In. *Found. Trends Hum.-Comput. Interact.* 14, 4 (nov 2021), 272–344. https://doi.org/10.1561/1100000083

[36] Aakanksha Naik, Abhilasha Ravichander, Norman M. Sadeh, Carolyn Penstein Rosé, and Graham Neubig. 2018. Stress Test Evaluation for Natural Language Inference. *ArXiv* abs/1806.00692 (2018). https://api.semanticscholar.org/CorpusID:46932607

[37] OpenAI. 2024. Text generation - OpenAI API. Retrieved from. https://platform.openai.com/docs/guides/text-generation

[38] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL]

[39] Neoklis Polyzotis, Steven Whang, Tim Klas Kraska, and Yeounoh Chung. 2019. Slice Finder: Automated Data Slicing for Model Validation. In *Proceedings of the IEEE Int' Conf. on Data Engineering (ICDE), 2019.* https://arxiv.org/pdf/1807.06068.pdf

[40] Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic Prompt Optimization with "Gradient Descent" and Beam Search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 7957–7968. https://doi.org/10.18653/v1/2023.emnlp-main.494

[41] Christopher Ré, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. 2019. Overton: A data system for monitoring and improving machine-learned products. *arXiv preprint arXiv:1909.05372* (2019).

[42] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. https://arxiv.org/abs/1908.10084

[43] Marco Tulio Ribeiro and Scott Lundberg. 2022. Adaptive Testing and Debugging of NLP Models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 3253–3267. https://doi.org/10.18653/v1/2022.acl-long.230

[44] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 4902–4912. https://doi.org/10.18653/v1/2020.acl-main.442

[45] Paul Röttger, Bertie Vidgen, Dong Nguyen, Zeerak Waseem, Helen Margetts, and Janet Pierrehumbert. 2021. HateCheck: Functional Tests for Hate Speech Detection Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, Online, 41–58. https://doi.org/10.18653/v1/2021.acl-long.4

[46] Maarten Sap, Dallas Card, Saadia Gabriel, Yejin Choi, and Noah A. Smith. 2019. The Risk of Racial Bias in Hate Speech Detection. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 1668–1678. https://doi.org/10.18653/v1/P19-1163

[47] Burr Settles. 2009. Active learning literature survey. (2009).

[48] Ryan Smith, Jason Alan Fries, Braden Hancock, and Stephen H. Bach. 2022. Language Models in the Loop: Incorporating Prompting into Weak Supervision. *ArXiv* abs/2205.02318 (2022). https://api.semanticscholar.org/CorpusID:248524894

[49] Ian Sommerville. 2015. *Software Engineering* (10th ed.). Pearson.

[50] Taylor Sorensen, Joshua Robinson, Christopher Rytting, Alexander Shaw, Kyle Rogers, Alexia Delorey, Mahmoud Khalil, Nancy Fulda, and David Wingate. 2022. An Information-theoretic Approach to Prompt Engineering Without Ground Truth Labels. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 819–862. https://doi.org/10.18653/v1/2022.acl-long.60

[51] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615* (2022).

[52] Zhen Tan, Alimohammad Beigi, Song Wang, Ruocheng Guo, Amrita Bhattacharjee, Bohan Jiang, Mansooreh Karami, Jundong Li, Lu Cheng, and Huan Liu. 2024. Large Language Models for Data Annotation: A Survey. arXiv:2402.13446 [cs.CL]

[53] Xingchen Wan, Ruoxi Sun, Hootan Nakhost, Hanjun Dai, Julian Eisenschlos, Sercan Arik, and Tomas Pfister. 2023. Universal Self-Adaptive Prompting. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 7437–7462. https://doi.org/10.18653/v1/2023.emnlp-main.461

[54] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).

[55] Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, Zac Kenton, Sasha Brown, Will Hawkins, Tom Stepleton, Courtney Biles, Abeba Birhane, Julia Haas, Laura Rimell, Lisa Anne Hendricks, William Isaac, Sean Legassick, Geoffrey Irving, and Iason Gabriel. 2021. Ethical and social risks of harm from Language Models. arXiv:2112.04359 [cs.CL]

[56] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Weld Daniel S. 2019. Errudite: Scalable, Reproducible, and Testable Error Analysis. In *the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019).* https://www.aclweb.org/anthology/P19-1073.pdf

[57] Tongshuang Sherry Wu, Haiyi Zhu, Maya Albayrak, Alexis Axon, Amanda Bertsch, Wenxing Deng, Ziqi Ding, Bill Boyuan Guo, Sireesh Gururaja, Tzu-Sheng Kuo, Jenny T Liang, Ryan Liu, Ihita Mandal, Jeremiah Milbauer, Xiaolin Ni, N. Padmanabhan, Subhashini Ramkumar, Alexis Sudjianto, Jordan Taylor, Ying-Jui Tseng, Patricia Vaidos, Zhijin Wu, Wei Wu, and Chenyang Yang. 2023. LLMs as Workers in Human-Computational Algorithms? Replicating Crowdsourcing Pipelines with LLMs. *ArXiv* abs/2307.10168 (2023). https://api.semanticscholar.org/CorpusID:259982473

[58] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864* (2023).

[59] Chenyang Yang, Rishabh Rustogi, Rachel Brower-Sinning, Grace Lewis, Christian Kaestner, and Tongshuang Wu. 2023. Beyond Testers' Biases: Guiding Model Testing with Knowledge Bases using LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2023.* 13504–13519.

[60] Sohee Yang, Jonghyeon Kim, Joel Jang, Seonghyeon Ye, Hyunji Lee, and Minjoon Seo. 2024. Improving Probability-based Prompt Selection Through Unified Evaluation and Analysis. arXiv:2305.14877 [cs.CL]

[61] Xi Ye, Srinivasan Iyer, Asli Celikyilmaz, Veselin Stoyanov, Greg Durrett, and Ramakanth Pasunuru. 2023. Complementary Explanations for Effective In-Context Learning. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 4469–4484. https://doi.org/10.18653/v1/2023.findings-acl.273

[62] Peilin Yu and Stephen H. Bach. 2023. Alfred: A System for Prompted Weak Supervision. In *Annual Meeting of the Association for Computational Linguistics*. https://api.semanticscholar.org/CorpusID:258967373

[63] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, Article 437, 21 pages.

[64] Ruoyu Zhang, Yanzeng Li, Yongliang Ma, Ming Zhou, and Lei Zou. 2023. LLMaAA: Making Large Language Models as Active Annotators. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 13088–13103. https://doi.org/10.18653/v1/2023.findings-emnlp.872

[65] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate Before Use: Improving Few-shot Performance of Language Models. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 12697–12706.

[66] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models Are Human-Level Prompt Engineers. arXiv:2211.01910 [cs.LG]