Alfred: A System for Prompted Weak Supervision

Peilin Yu **Stephen H. Bach**
Department of Computer Science
Brown University
{peilin_yu, stephen_bach}@brown.edu

Abstract

Alfred is the first system for programmatic weak supervision (PWS) that creates training data for machine learning by prompting. In contrast to typical PWS systems where weak supervision sources are programs coded by experts, Alfred enables users to encode their subject matter expertise via natural language prompts for language and vision-language models. Alfred provides a simple Python interface for the key steps of this emerging paradigm, with a high-throughput backend for large-scale data labeling. Users can quickly create, evaluate, and refine their prompt-based weak supervision sources; map the results to weak labels; and resolve their disagreements with a label model. Alfred enables a seamless local development experience backed by models served from self-managed computing clusters. It automatically optimizes the execution of prompts with optimized batching mechanisms. We find that this optimization improves query throughput by 2.9× versus a naive approach. We present two example use cases demonstrating Alfred on YouTube comment spam detection and pet breeds classification. Alfred is open source, available at https://github.com/BatsResearch/alfred.

1 Introduction

Acquiring labeled data is a significant challenge for machine learning for its time-consuming and expensive nature. Programmatic weak supervision (PWS) provides a more efficient method of data annotation by using noisy heuristics to label data. In a typical PWS setup, domain experts design labeling functions (LFs) as programs that vote for a label or abstain. (Ratner et al., 2016, 2017) Recently, there has been a growing interest in creating LFs from large, pre-trained models through prompting (Smith et al., 2022; Arora et al., 2022; Zhang et al., 2022b). In the shift to this new setting, executing LFs goes from the least to the most computationally expensive part of the process, highlighting the importance of providing a software infrastructure that facilitates efficient development. However, existing toolkits for large language models mainly prioritize prompt templating and tuning, leaving an unmet need for a system that that connects prompting with the creation of training data.

Prompted models offer a unique opportunity to enhance existing PWS systems. Traditional PWS systems require programming LFs with code that specifies heuristic domain knowledge. With large pre-trained models, natural language-based prompts can be used as LFs, also known as prompted LFs (Smith et al., 2022). This approach allows the easy expression of complex rules that were previously difficult to specify using code, as the example in Figure 1 shows. The ability to use prompts to define labeling functions simplifies and streamlines the weak supervision process, as well as potentially elevating the quality of the annotations. This benefit is particularly helpful for tasks involving computer vision, where previously PWS has been limited to tasks for which models can identify key features (such as objects) over which to program rules. Whether the domain is language or multi-modal, prompts let users experiment with different heuristics (and phrasings of those heuristics). Therefore, enabling an iterative development experience is essential for the success of a prompt-based PWS system.

The switch to prompted models for weak supervision also presents significant challenges. It first requires rethinking the abstractions and workflow of first-generation PWS systems. Instead of editing code and managing libraries of functions, users must manage libraries of prompts, track their outputs on multiple datasets, and develop strategies for mapping those outputs to labels. This change is complicated by large models' demand for computational resources. The throughput of the models is a new development bottleneck. The extra overhead of remotely hosted models is a further hindrance to

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 3: System Demonstrations, pages 479–488

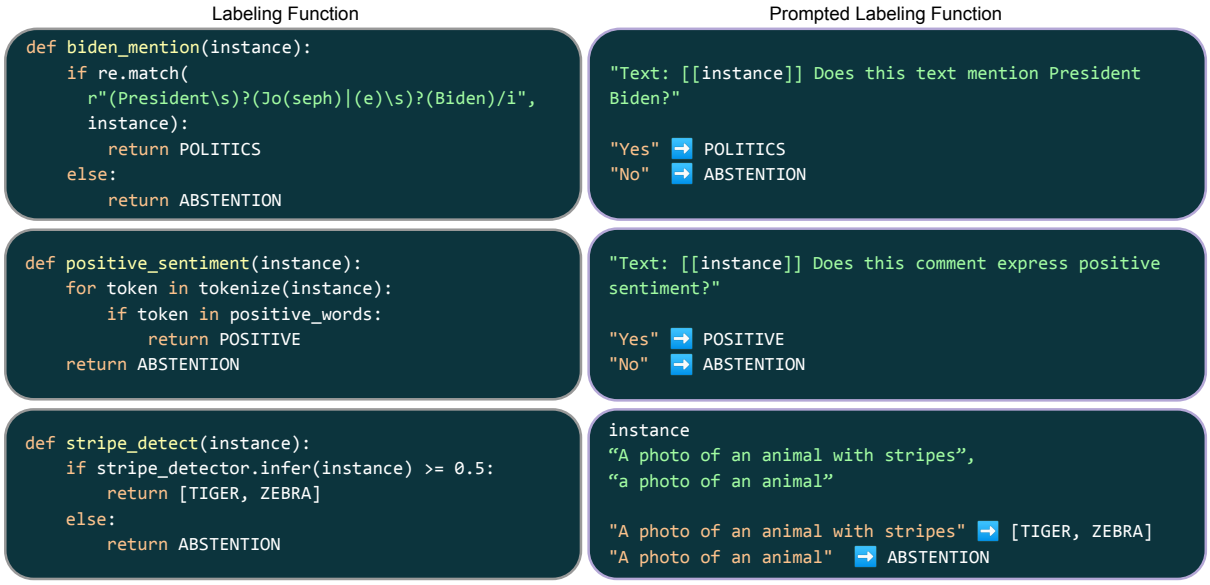

Figure 1: Examples of a labeling function versus a prompted labeling function. For the first example, each expresses supervision relating mentions of President Biden to the category of politics. Instead of specifying an intricate regular expression, a prompted labeling function uses the prompt “Text: [[instance]] Does this text mention President Biden?” where [[instance]] is replaced by the news article to be labeled. The response is mapped to a vote on the true label. The second example demonstrates how heuristics about positive sentiment that were previously hard to define can be flexibly expressed as a natural language question. Instead of defining a set of keywords for fuzzy sentiment matching, we can simply ask large pretrained models for answers about the sentiment. For the third example, we consider an animal labeling task where we use the visual attributes “stripes” to vote for the classes TIGER and ZEBRA. Previously, we would need to first collect supervised training data for attributes like stripes and then train classifiers to make the decisions. With modern vision-language models (e.g. CLIP), we can simply express the attribute detection task as a set of candidate prompts.

the iterative workflow of weak supervision.

Existing open-source software for prompting concentrates on prompt engineering (Orr, 2022; Bach et al., 2022), prompt chains (Chase, 2022) or continuous (i.e., soft) prompt tuning (Ding et al., 2022); placing less emphasis on throughput for a large-model-in-the-loop workflow. Additionally, many existing open-source systems have not developed support for vision-language models, despite their benefits for data labeling. Incorporating large pre-trained models into a PWS system remains an open problem in the open-source software space, requiring innovative solutions to address these challenges and complement existing software focused on other aspects of prompting.

We present Alfred, a versatile PWS system that leverages large pre-trained models for image and text annotations. Alfred aims to provide an environment for the rapid development of prompt-based supervision, while maintaining a consistent development experience similar to established PWS systems. We designed Alfred with usability and efficiency in mind, aiming to provide a rapid and smooth experience for developing prompt-based supervision. Alfred supports popular large language models from Hugging Face’s transformer package (Wolf et al., 2020), including the GPT family (Radford et al., 2019), the T5 family (Raffel et al., 2020), etc., and vision-language models like CLIP (Radford et al., 2021), etc. Alfred also supports local ONNX models, or API-based models from OpenAI, AI21, and Cohere. Moreover, Alfred provides easy templating tools to help users quickly create, evaluate, and refine prompted LFs. Alfred offers easy ways to deploy inference servers remotely, in addition to local model hosting. Alfred also optimizes model inference throughput with improved batching techniques and provides utilities for efficient LLM deployment and interaction. Finally, Alfred contains a library of label models to distill the outputs of prompted labeling functions into the final training datasets for downstream end models.

Alfred is a prototype for a second generation of PWS systems with prompting at their core. To this end, we highlight three key feature of Alfred:

- **Prompt-based weak supervision for images**

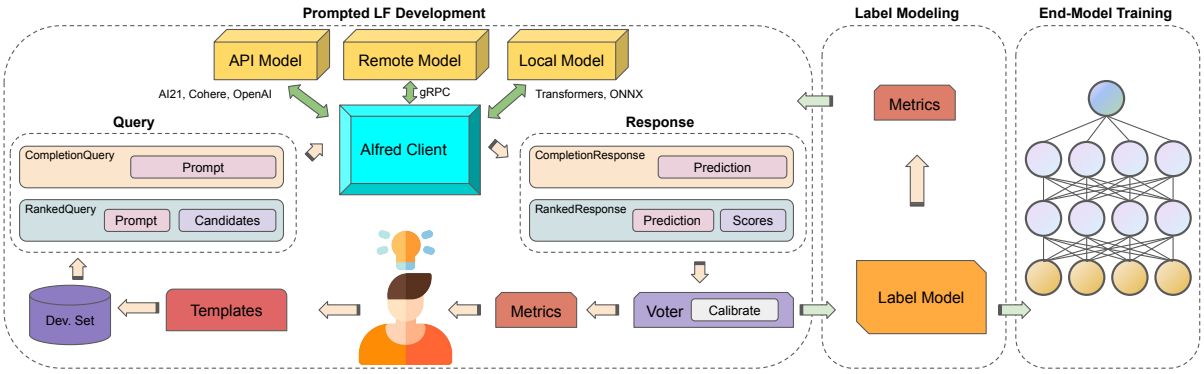

Figure 2: A typical workflow for programmatic weak supervision with Alfred. First, developers use Alfred to iteratively design, evaluate, and refine their prompted labeling functions (LFs). They use prompt Templates to generate Queries to Models based on data. The responses of Models are mapped to votes on the true labels for examples by Voters, which can be calibrated. Then the included Label Models combine the noisy votes to produce probabilistic training labels for an end model.

and text. Alfred provides the necessary tools for users to create, evaluate, and refine prompt templates for both image and text weak supervision tasks. The inclusion of a query caching system that automatically stores and updates model responses facilitates development.

- **Optimized inference throughput.** Alfred implements a dynamic batching mechanism that optimizes large sets of prompts. This feature allows models hosted by Alfred to achieve 2-3× greater throughput than naive implementations.
- **Seamless local development experience.** Alfred can host models remotely and make them accessible to developers via gRPC, a high-throughput protocol for remote procedure calls.[1] It enables sending datasets to servers in large chunks to maintain higher throughput. Alfred also implements a SSH-based port-forwarding utility for the gRPC connection, easing deployment on shared clusters such as those at universities.

2 Related Work and Background

Alfred sits at the intersection of programmatic weak supervision and large pretrained models. In this section, we overview the most related work.

Programmatic Weak Supervision. Traditionally, supervised learning relied on manually labeled data, and data labeling has been seen as a key bottleneck for many applications. Recently, a family of programmatic weak supervision (PWS) methods have offered an alternative to costly manual annotations by incorporating multiple sources of noisy labels to create training datasets (Zhang et al., 2022a). Typically, a PWS system such as Snorkel (Ratner et al., 2017) has a three-stage setup: First, developers will create heuristics called labeling functions (LFs) that vote on the label for an example (or abstain). Second, a label model will reconcile the noisy votes and provide probabilistic labels for the data. Finally, freshly annotated data is used to train an end model with a noise-aware loss objective (e.g. in a classification setting, this can be a soft cross entropy (Ratner et al., 2016)). Alfred focuses on the first two stages and aim to efficiently incorporate modern large pretrained models into the development workflow.

Prompting for Pre-Trained Models. With the emergence of large pre-trained langugae and vision-language models, prompting has become a popular approach to many few-shot and zero-shot tasks (Brown et al., 2020; Schick and Schütze, 2021a; Radford et al., 2021). Prompting can create training examples, generate data, modify datasets, and improve model reasoning (Schick and Schütze, 2021b; Ye et al., 2022; Chia et al., 2022; Wu et al., 2022; Wang et al., 2022; Wei et al., 2022; Zelikman et al., 2022). This presents a unique opportunity for combining prompting for large pretrained models and weak supervision. Recent studies have investigated strategies to combine large language models into weak supervision frameworks (Smith et al., 2022; Chen et al., 2022; Arora et al., 2022; Zhang et al., 2022b). Alfred aims to provide a platform for the rapid development of weak supervision applications that rely on large pre-trained language and vision-language models, as well as enable experimentation with new ways of using those models.

[1] grpc.io

Systems for Prompt Development. Prompting has led to the development of software toolkits that aid in prompting and research across various tasks. Many tools have been developed for various use cases with large language models. PromptSource (Bach et al., 2022) is a development environment for creating and archiving sets of prompts. OpenPrompt (Ding et al., 2022) is a library focused on tuning prompts and prompted models with training data. Manifest (Orr, 2022) provides a unified front end for prompting large language models via different APIs. LangChain (Chase, 2022) provides convenient utilities for building applications that chain together multiple prompts and outputs. To complement the existing tools in this growing space, Alfred is designed to be a PWS system based on prompting both large language and vision-language models.

3 Prompted LF Development

Alfred is designed to enable the development and application of prompted labeling functions (LFs) Compared to the typical workflow of PWS systems, where developing LFs is not computationally demanding, developing prompted LFs has model inference as a bottleneck. These large models are often hosted remotely on virtual instances or computing clusters, which can add to the challenge of iterative prompt development. In an iterative development environment, creating prompted labeling functions requires a platform that provides optimal throughput and low latency for a rapid local development experience. To illustrate Alfred's key focuses, we illustrate a typical workflow (Figure 2) for using Alfred to create a training dataset:

Step 1: Task Setup For a large model to be used in the development loop, developers can elect to use either self-hosted models or API-based models. For self-hosted models, Alfred provides an `AlfredServer` to host the model on cloud or cluster nodes. As the main development interface, the user can simply start a `Client` by specifying the type of model. Before creating prompted LFs, users need to familiarize themselves with the task by exploring the raw, unlabeled dataset. If there is no development subset available, the developer can annotate a small portion of the data as a held-out evaluation benchmark. Alfred implements a `Dataset` class based on Apache Arrow for fast data access. User may load a `Dataset` from CSV, JSON or Dataframe objects. It also offers direct support for datastes from the Hugging Face 'Dataset' package (Lhoest et al., 2021). During the exploration process, developers may gain insights into the data and the label space, and identify potentially useful heuristics. Moreover, users can freely experiment with prompts with a few unlabeled instances by directly interacting with the `Client`.

Step 2: Iterative Prompt Development: When the user is ready for prompt development, they can use a `Template` to define a prompted LF for either text completion or scoring schemes. A `Template`, given a `Dataset`, will produce the corresponding `Query` objects. `Client` will return the appropriate `Response` object for each `Query`. To map the model responses to votes, users define the corresponding `Voter` and identify the label maps and matching functions to be used for each prompted LFs. Label maps define how potential model responses are associated with the label space. Matching functions specify how the `Voter` determines a match. By default, Alfred employs an exact match mechanism, but this can be substituted with user-defined matching functions for uncased matching or embedding similarity matching, etc. A `Voter` can be optionally calibrated to reduce model-specific biases (Zhao et al., 2021). With the model responses and the Voter, users can obtain the label votes for each of their prompted LFs and examples in a matrix format. Finally, users can evaluate the quality of their prompted LFs using a set-aside development `Dataset` with desired metrics. Here, users can continue to refine their prompted LFs and iterate as necessary. Once users are satisfied with the performance of their development benchmark, they may proceed.

Step 3: Aggregate Prompt Responses Finally, Alfred can aggregate the votes from each `Voter` with a `LabelModel` to produce probabilistic estimates of the true labels for the examples. Alfred also supports *partial labels*, i.e., labels that narrow down the possible set of classes but are not specific enough to vote on a single class (Yu et al., 2022). The probabilistic labels can then be used to train a wide range of end models.

4 System Design

In this section, we describe and highlight the key design decisions for Alfred.

4.1 Query and Response Types

We identify two main patterns using prompts

```
from alfred import Client
from alfred.fm.query import RankedQuery,
                            CompletionQuery
LMClient = Client(...)

headline = "Liverpool wins 7-0!"

LMClient(
    CompletionQuery(headline
    + " What is the topic of this headline?")
)
# Example Response:
# >> CompletionResponse(prediction="Football")

LMClient(
    RankedQuery(headline
    + " What is the topic of this headline?",
      candidate=["Sports", "Politics",
                 "Tech", "Business"])
)
# Example Response:
# >> RankedResponse(prediction="Sports",
#      scores={"Sports":0.76, "Politics":0.10,
#              "Tech":0.07, "Business":0.07 })
```

Figure 3: Typed `Query` and `Response` in Alfred

for PWS: text completion and scoring. Text completion is when a language model generates responses using a heuristic decoding strategy over the whole model vocabulary, while scoring is when a language ranks candidate completions or a vision-language model ranks candidate prompts, i.e., captions. Alfred implements typed `Query` and `Response` classes for these two patterns (Figure 3). Upon applying the `Template` operation on a dataset instance, it produces either a `CompletionQuery` or a `RankedQuery` for each instance based on the `Template` definition. The resulting query can be directly fed into the `Client`. The `Client` then returns a corresponding `CompletionResponse` or `RankedResponse` with the prediction as the main payload, along with any other requested or useful information, such as the logits for each candidate.

4.2 Templates for Prompted LFs

Prompt templates are at the core of systems for prompting. In Alfred, prompt templates are expressed as `Template` objects. For natural language tasks, users use the `StringTemplate` class. To produce `Query` objects, users can call 'Template.apply(instance)' or 'Template.apply_to_dataset(dataset).' A `StringTemplate` is defined with a template string with keywords enclosed by double square brackets, e.g. "[[text]] Does the previous context express spouse relation between [[entity_a]] and [[entity_b]]?". An optional field for `Template` is 'answer_choices,' where one may specify the candidate completions. By specifying the 'answer_choices,' the `StringTemplate` would yield a `RankedQuery`. An example code snippet showing the creation of a `RankedQuery` is in Figure 4. For image annotation tasks, users may define an `ImageTemplate` by specifying the candidate prompts. Upon applying to images, `ImageTemplate` will produce `RankedQuery` objects with the images and candidate prompts.

4.3 Throughput Optimization

Alfred is designed to handle large numbers of queries. Self-hosted models from the Transformers package (Wolf et al., 2020) are set up to use model parallelization enabled by Accelerate (Sylvain Gugger, 2022), with user-customizable device maps for parallelizing the model across multiple GPUs. Alfred adopts a dynamic batching strategy that groups instances with similar lengths together and adjusts the input batch size dynamically to maximize model inference throughput. The core idea of the dynamic batching strategy is to group input instances with similar token lengths to minimize padding and maximize memory utilization.

With the dynamic batching strategy, on a node with 8 NVIDIA Tesla V100s, Alfred achieves a speedup of up to 2.5× and a token throughput increase of 2.9× for approximately 500 queries (∼21,000 tokens) compared to an unoptimized strategy with T0++ (Sanh et al., 2022), an 11-billion parameter T5-based (Raffel et al., 2020) language model in FP32. Additionally, Alfred includes a client-side query caching system that automatically stores and updates model responses to facilitate prompt development and avoid redundant queries during development. Alfred also implements a server-side caching system for large multi-modal pretrained models such as CLIP. At inference time, Alfred will cache the input data with its corresponding encoded latent representations from different encoder head for each modalities. This server-side caching system effectively avoids redundant encoding computation on the server end.

```python
from alfred.template import StringTemplate, ImageTemplate

string_template = StringTemplate(
    "Context: [[text]]\n\nIs the above text about weather?", answer_choices = ["Yes", "No"]
)
example = {'text': "A pleasant day with a sunny sky."}
prompt = string_template.apply(example)
# >> RankedQuery("Context: A pleasant day with a sunny sky.\n\nIs the above text about weather?",
#                candidates=["Yes", "No"])

image_template = ImageTemplate(
    {"label": ["cat", "dog"]},
    template = "A photo of [[label]]."
)
example = cat_image
prompt = image_template.apply(example)
# >> RankedQuery(example, candidates=["A photo of cat.", "A photo of dog."])
```

Figure 4: Example code snippet for creating a `RankedQuery` from a `StringTemplate` or a `ImageTemplate`.

4.4 Remote Self-Hosting of Models

The computational demands of large pre-trained models can pose a challenge when using them for weak supervision development. To address this challenge, Alfred provides utilities for deploying and interacting with models on remote virtual instances or computing clusters. Additionally, Alfred implements a SSH-based tunneling service that ensures a secure local connection while preserving all Alfred functionality. The tunneling utility also simplifies deployment of the server on shared computing clusters, with the login node serving as a jump host for the computing node. This is particularly useful for using Alfred on centrally-managed shared computing clusters such as those at universities. Alfred's built-in SSH tunneling is also capable of handling 2-factor authentication, which is common for shared clusters. To enable efficient communication between the client and server, Alfred uses gRPC, a high-performance, open-source remote procedure call framework. This enables Alfred to provide a seamless development experience for weak supervision development without the need for expensive local resources.

4.5 Mapping Responses to Votes

Another core piece of the Alfred system is the `Voter` class. Each `Voter` defines how to map model responses to votes for the true label of an exmaple. The votes can be class labels or partial labels (e.g., attributes) specified by the users. The voting mechanism also relies on a matching function, which by default only casts a vote for an exact match. Users may provide their intended matching mechanisms such as uncased matching or embedding similarity matching for more flexibility for each `Voter`. Furthermore, `Voter` can be contextually calibrated for the specific `Template` class to reduce model bias towards predicting certain answers. Recent studies show calibration can be helpful for many prompt-based tasks (Zhao et al., 2021; Smith et al., 2022). By calling 'Client.calibrate(Template, Voter),' Alfred will calibrate the voting weights according to the strategy proposed by Zhao et al. and automatically apply the calibration during voting.

4.6 Label Models for Aggregating Votes

Alfred currently includes four label models for combining the votes from prompted labeling functions. The four label models are available to meet different use cases. The `MajorityVote` model is a baseline option suitable for fast development iteration, while the `NaiveBayes` model is recommended as the standard label model. Alfred also includes `NPLM` (Yu et al., 2022) (noisy partial label model) to support weak supervision from partial labels, which are labels that narrow down the possible set of classes but are not specific enough to vote on a single class. `FlyingSquid` (Fu et al., 2020) is the fourth model option and is recommended when `MajorityVote` is not accurate enough but more speed than `NaiveBayes` is needed. These label model classes have a unified interface, providing a consistent experience for users. After processing votes, the label model module generates probabilistic labels, represented as a distribution over the label space, for the given unlabeled dataset. Finally users can use the estimated probabilistic labels to train an end model for the downstream task.

5 Example Use Cases

In this section, we present two example use cases for how Alfred can be used to create training data for specific machine learning tasks using natural language prompts in both text and image domains. We measure the labeling quality by taking the top-1 accuracy of the estimated probabilistic labels given by the label model. The notebooks to reproduce these examples are in the Alfred repository.

5.1 Youtube Comment Spam Detection

Zero Shot	Prompted LFs	Prompted LFs+C
46.8	57.8	65.3

Table 1: Top-1 accuracy on YouTube spam detection. Zero Shot refers to prompting T0++ directly. +C means applying contextual calibration on the `Voter` objects.

In this experiment, we use Alfred to annotate the training split of YouTube spam detection dataset. (Alberto et al., 2015) We replicate the setup used by Smith et al. The prompts are translated from the code-based labeling functions provided by the WRENCH benchmark (Zhang et al., 2021), a comprehensive weak supervision benchmark. Alfred also includes a `WrenchBenchmarkDataset` abstraction for easily running this benchmark. In total, we define 10 prompted labeling functions with `StringTemplate` objects. Responses are mapped to votes using `Voter` objects. For this experiment, we use T0++ (Sanh et al., 2022) as the backbone model for Alfred. Following Smith et al., we also calibrate the responses from T0++ when voting using the contextual calibration strategy proposed by Zhao. Finally we aggregate the votes using the `NaiveBayes` label model to produce the probablistic labels. Table 1 shows that Alfred makes reproducing the results of Smith et al. easy, demonstrating that the combination of weak supervision and calibration yield a dramatic improvement over zero-shot prompting alone.

5.2 Pet Breed Classification

Zero Shot	Prompted LFs
86.0	92.4

Table 2: Top-1 accuracy on Oxford-IIIT Pet breed classification. Zero Shot refers to prompting CLIP directly.

Traditionally, programmatic weak supervision for vision has been limited by the ability to express supervision in code, relying on models such as object or attribute detectors to extract features and classify. However, these object detectors often depend heavily on supervised training data, becoming a bottleneck for applying programmatic weak supervision in various vision tasks. Fortunately, with large-pretrained vision-language model like CLIP (Radford et al., 2021), we are now able to express supervision with natural language. For this task, we develop prompts to classify 37 different breeds of pets from the Oxford-IIIT Pet dataset (Parkhi et al., 2012). We use CLIP-ViT/L-14 as the backbone model and developed three simple prompted LFs. The first two prompted LFs use templates "a photo of [[label]]" and "a photo of [[label]] [cat/dog]" where "[cat/dog]" is selected based on the breed. The third prompted LF produces a partial label with the template "a photo of [cat/dog]", encouraging fine-grained labels to match with the coarse-grained type detected by CLIP. We combine the votes using the `NPLM` label (Yu et al., 2022) to support weak supervision at various levels of granularity in the label space. Table 2 shows that this multi-granular weak supervision provides a nice boost over zero-shot prompting alone.

6 Discussion and Future Work

This paper introduces Alfred, a prototype for the next generation of programmatic weak supervision systems that leverage the potential of large pretrained models. Alfred complements the existing ecosystem of large-pretrained-model toolkits, offering optimized inference throughput, a smooth local development experience, and compatibility with vision-language models to support image annotation tasks. Alfred represents a notable advancement in the domain of programmatic weak supervision, as it enables users to express their domain-specific knowledge and heuristics with flexible natural language prompts for language and vision-language models. This approach can be more user-friendly than conventional PWS systems, which requires expert programming of weak supervision sources. Our objective is for Alfred to serve as the infrastructure and experimentation platform for many future weak supervision research projects and applications. Furthermore, we plan to extend Alfred's capabilities to accommodate a wider range of multimodal large pre-trained models, such as Whisper (Radford et al., 2022) and LayoutLMs (Xu et al., 2020b,a; Huang et al., 2022).

Limitations

Alfred is a prototype for the second generation of PWS systems, which incorporate large pre-trained models. However, there are some potential limitations to consider. As with all PWS approaches, application quality is limited by the quality of the weak supervision sources used to vote on the labels. In this case of prompted labeling functions, this depends on how well suited the prompts and model are to the task and domain. If they are not well suited, then additional fine-tuning of the prompted models will be necessary. Compared with traditional labeling functions written in code, understanding when and why labeling functions fail on certain examples can be particularly challenging. Methods for explanations such as minimal contrastive edits (Ross et al., 2021) can potentially help address this limitation. We plan to explore incorporating such methods into Alfred.

Ethics Statement

One major concern for Alfred is the potential for biased or unfair labeling. Large pre-trained models are trained on massive datasets, which can reflect societal biases and inequalities. Consequently, supervision generated by these models can perpetuate and amplify these biases, leading to discrimination or unfair treatment in downstream applications. Therefore, it is essential to carefully consider the quality and representativeness of the backbone model for Alfred, as well as the prompts used for labeling data. To address potential labeling biases, human oversight and auditing are needed during the development loop to spot and correct any issues. While Alfred has the potential to enhance the efficiency of programmatic data labeling, it is crucial to carefully consider and address potential ethical challenges.

Acknowledgments

We appreciate the helpful comments and discussion with Andrew Yuan, Avi Trost, Nihal Nayak and Zheng-Xin Yong. This material is based on research sponsored by Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL) under agreement number FA8750-19-2-1006. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL) or the U.S. Government. We gratefully acknowledge support from Google and Cisco. Disclosure: Stephen Bach is an advisor to Snorkel AI, a company that provides software and services for weakly supervised machine learning.

References

Túlio C Alberto, Johannes V Lochter, and Tiago A Almeida. 2015. Tubespam: Comment spam filtering on youtube. In *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*, pages 138–143. IEEE.

Simran Arora, Avanika Narayan, Mayee F Chen, Laurel J Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. 2022. Ask me anything: A simple strategy for prompting language models. *arXiv preprint arXiv:2210.02441*.

Stephen H. Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged S. Al-shaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Xiangru Tang, Mike Tian-Jian Jiang, and Alexander M. Rush. 2022. Promptsource: An integrated development environment and repository for natural language prompts.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Harrison Chase. 2022. LangChain.

Mayee F Chen, Daniel Y Fu, Dyah Adila, Michael Zhang, Frederic Sala, Kayvon Fatahalian, and Christopher Ré. 2022. Shoring up the foundations: Fusing model embeddings and weak supervision. In *Uncertainty in Artificial Intelligence*, pages 357–367. PMLR.

Yew Ken Chia, Lidong Bing, Soujanya Poria, and Luo Si. 2022. Relationprompt: Leveraging prompts to generate synthetic data for zero-shot relation triplet extraction. *arXiv preprint arXiv:2203.09101*.

Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Haitao Zheng, and Maosong Sun. 2022. OpenPrompt: An open-source framework for prompt-learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 105–113, Dublin, Ireland. Association for Computational Linguistics.

Daniel Y. Fu, Mayee F. Chen, Frederic Sala, Sarah M. Hooper, Kayvon Fatahalian, and Christopher Ré. 2020. Fast and three-rious: Speeding up weak supervision with triplet methods. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*.

Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. 2022. Layoutlmv3: Pre-training for document ai with unified text and image masking. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 4083–4091.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. 2021. Datasets: A community library for natural language processing. *arXiv preprint arXiv:2109.02846*.

Laurel Orr. 2022. Manifest. `https://github.com/HazyResearch/manifest`.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. 2012. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 11, page 269. NIH Public Access.

Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29.

Alexis Ross, Ana Marasović, and Matthew Peters. 2021. Explaining NLP models via minimal contrastive editing (MiCE). In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3840–3852, Online. Association for Computational Linguistics.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

Timo Schick and Hinrich Schütze. 2021a. Few-shot text generation with natural language instructions. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 390–402, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021b. Generating datasets with pretrained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6943–6951, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ryan Smith, Jason A Fries, Braden Hancock, and Stephen H Bach. 2022. Language models in the loop: Incorporating prompting into weak supervision. *arXiv preprint arXiv:2205.02318*.

Thomas Wolf Philipp Schmid Zachary Mueller Sourab Mangrulkar Sylvain Gugger, Lysandre Debut. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. `https://github.com/huggingface/accelerate`.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Yuxiang Wu, Matt Gardner, Pontus Stenetorp, and Pradeep Dasigi. 2022. Generating data to mitigate spurious correlations in natural language inference datasets. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2660–2676, Dublin, Ireland. Association for Computational Linguistics.

Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, et al. 2020a. Layoutlmv2: Multi-modal pre-training for visually-rich document understanding. *arXiv preprint arXiv:2012.14740*.

Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020b. Layoutlm: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1192–1200.

Jiacheng Ye, Jiahui Gao, Qintong Li, Hang Xu, Jiangtao Feng, Zhiyong Wu, Tao Yu, and Lingpeng Kong. 2022. Zerogen: Efficient zero-shot learning via dataset generation. *arXiv preprint arXiv:2202.07922*.

Peilin Yu, Tiffany Ding, and Stephen H. Bach. 2022. Learning from multiple noisy partial labelers. In *Artificial Intelligence and Statistics (AISTATS)*.

Eric Zelikman, Yuhuai Wu, and Noah D Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*.

Jieyu Zhang, Cheng-Yu Hsieh, Yue Yu, Chao Zhang, and Alexander Ratner. 2022a. A survey on programmatic weak supervision. *arXiv preprint arXiv:2202.05433*.

Jieyu Zhang, Yue Yu, Yinghao Li, Yujing Wang, Yaming Yang, Mao Yang, and Alexander Ratner. 2021. WRENCH: A comprehensive benchmark for weak supervision. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Rongzhi Zhang, Yue Yu, Pranav Shetty, Le Song, and Chao Zhang. 2022b. Prboost: Prompt-based rule discovery and boosting for interactive weakly-supervised learning. *arXiv preprint arXiv:2203.09735*.

Fang Zhao. 2022. Auto-correction dans un analyseur neuronal par transitions : un comportement factice ? (self-correction in a transition-based neural parser : a spurious behaviour ?). In *Actes de la 29e Conférence sur le Traitement Automatique des Langues Naturelles. Volume 2 : 24e Rencontres Etudiants Chercheurs en Informatique pour le TAL (RECITAL)*, pages 20–32, Avignon, France. ATALA.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning*, pages 12697–12706. PMLR.