A DECLARATIVE SYSTEM FOR OPTIMIZING AI WORKLOADS

1 Introduction

Advances in AI models have driven progress in applications such as question answering [64], chatbots [13], autonomous agents [46, 51], and code synthesis [33, 24]. In many cases these systems have evolved far beyond posing a simple question to a chat model: they are complex AI systems [63] that combine elements of data processing, such as Retrieval Augmented Generation (RAG); ensembles of different models; multi-step chain-of-thought reasoning; and in many cases, cloud-based modules.

It is easy for the runtime, cost, and complexity of these AI systems to escalate quickly, particularly when applied to large collections of documents. Consider a few simple AI-powered analytical tasks:

• Real Estate Search (Figure 2b) — In this use case, a homebuyer wants to use online real estate listing data to find a place that is (a) modern and attractive, and (b) within two miles of work. Test (a) is a semantic search task that possibly involves analyzing images, while (b) is a more traditional distance calculation over extracted geographic data. Any implementation needs to process a large number of images and listings, limit its use of slow and expensive models, and still obtain high-quality results.

These tasks:

1. Interleave traditional data processing with AI-like semantic reasoning

2. Are data-intensive: each source dataset could reasonably range from hundreds to millions of records

3. Can be decomposed into an execution tree of distinct operations over sets of data objects 4. May result in answers of varying quality

Semantic Analytics Applications: Taken together, these criteria outline a broad class of AI programs that are important, complex, and potentially very optimizable; we call them *semantic analytics applications* — or, SAPPs. We believe there is a large set of such use cases that mix conventional data analytics with transformations and filters that would not be possible without AI methods. Such workloads frequently require interleaved data acquisition steps, conventional analytical queries, and AI operations. The AI operations process unstructured data, require broad domain knowledge to implement, or have specifications that users may not be able to implement correctly with traditional source code.

Challenges: Naively scaling AI systems to process SAPPs with thousands or millions of inputs appears to require spending a huge amount of runtime and money executing high-end AI models. The performance gap between traditional data processing components and AI-powered components is profound. For example, a high-quality open-source LLM running on a modern GPU might process 100-125 tokens per second. Assuming a token is represented by 5 bytes (on average), such a model yields a throughput of *less than 1 KB per second*. OpenAI’s new GPT-4o model currently costs 5 USD for 1M input tokens, or in other words 5 USD for processing just 5MB of data. These numbers are many orders of magnitude worse than any other component of the modern data processing stack, such as data storage, network bandwidth, SQL query processing time, and so on.

At its core, optimizing an AI system to execute a given SAPP workload requires making accurate predictions about the runtime, cost, and quality of each semantic and conventional data processing step. Estimating these metrics for semantic tasks can be particularly challenging. For example, estimating the runtime and cost for a vision model requires knowing the average number of input and output tokens per record as well as the total number of records that will be processed by the model (i.e., its cardinality). Estimating the quality of an output — especially without labelled data — may require using heuristics that can be error prone, or comparing against an expensive "champion" model. Finally, for every physical optimization available to the system (such as using an ensemble of vision models or decreasing the image resolution), the optimizer needs to predict the optimization’s impact on these metrics.

1 **import palimpzest as pz**

2

3 **class Email**(pz.TextFile):

4 *"""Represents an email, which can subclass a text file"""*

5 sender = pz.StringField(desc="The email address of the sender", required=**True**) 6 subject = pz.StringField(desc="The subject of the email", required=**True**) 7

8 *# define logical plan*

9 emails = pz.Dataset(source="enron-emails", schema=Email) *# invokes a convert operation* 10 emails = emails.filter("The email is not quoting from a news article or an article ...") 11 emails = emails.filter("The email refers to a fraudulent scheme (i.e., **\"**Raptor**\"**, ...") 12

13 *# user specified policy*

14 policy = pz.MinimizeCostAtFixedQuality(min\_quality=0.8)

15

16 *# execute plan*

17 results = pz.Execute(emails, policy=policy)

Figure 3: The AI program written using PALIMPZEST for the Legal Discovery workload.