Databases Unbound: Querying All of the World's Bytes with AI

Samuel Madden
MIT CSAIL
madden@csail.mit.edu

Michael Cafarella
MIT CSAIL
michjc@csail.mit.edu

Michael Franklin
Univeristy of Chicago
mjfranklin@uchicago.edu

Tim Kraska
MIT CSAIL
kraska@csail.mit.edu

ABSTRACT

Over the past five decades, the relational database model has proven to be a scaleable and adaptable model for querying a variety of structured data, with use cases in analytics, transactions, graphs, streaming and more. However, most of the world's data is unstructured. Thus, despite their success, the reality is that the vast majority of the world's data has remained beyond the reach of relational systems.

The rise of deep learning and generative AI offers an opportunity to change this. These models provide a stunning capability to extract semantic understanding from almost any type of document, including text, images, and video, which can extend the reach of databases to all the world's data. In this paper we explore how these new technologies will transform the way we build database management software, creating new that systems that can ingest, store, process, and query all data. Building such systems presents many opportunities and challenges. In this paper we focus on three: scalability, correctness, and reliability, and argue that the declarative programming paradigm that has served relational systems so well offers a path forward in the new world of AI data systems as well. To illustrate this, we describe several examples of such declarative AI systems we have built in document and video processing, and provide a set of research challenges and opportunities to guide research in this exciting area going forward.

And lovely apparitions,–dim at first,
Then radiant, as the mind arising bright
From the embrace of beauty (whence the forms
Of which these are the phantoms) casts on them
The gathered rays which are reality–
Shall visit us the progeny immortal
Of Painting, Sculpture, and rapt Poesy,
And arts, though unimagined, yet to be;
Prometheus Unbound, Percy Bysshe Shelley

PVLDB Reference Format:
Samuel Madden, Michael Cafarella, Michael Franklin, and Tim Kraska. Databases Unbound: Querying All of the World's Bytes with AI. PVLDB, 17(12): 4546-4554, 2024.
doi:10.14778/3685800.3685916

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685916

1 INTRODUCTION

Database systems are the dominant paradigm for querying structured data, but most of the world's data – especially with the rise of the Internet, digital photos, videos, high rate sensing equipment, and so on – is not structured as tables that are readily accessible to databases. Over the past few years, however, new AI models that are capable of remarkable "understanding" of this unstructured data have arisen, and potentially provide a way to extend the power of querying to all data; for example, LLMs can effectively answer arbitrary queries over text documents and multi-modal vision models can answer similar questions about images.

In this paper, we argue for a new generation of database systems that is "unbound" from the tabular/relational restriction through its use of these new AI models to query *all* data. Such a system has the potential to power many exciting enterprise applications over data. Imagine, for example:

- A data engineer who can quickly transform all financial documents, including pay stubs, financial reports, loan applications, public filings, prospectuses, and more, into queryable documents and structured tables for analysis.
- A lawyer who can quickly search hundreds of thousands of pages of documents in legal discovery to look for particular phrases, names, or general topics, and tabulate statistics about the prevalence of each.
- A medical researcher who can quickly summarize all of the data about patients with a particular disease in their hospital, and compare their findings to the known findings in the medical literature.
- An investment banker who can quickly correlate stock data with news announcements and tweets from a set of important influencers.
- An AI researcher who can quickly extract the performance of recent competitor models from figures in PDFs on arXiv.

These tasks span familiar data processing tasks – including retrieval (i.e., finding records that match a criteria), extraction (i.e., creating structured records from unstructured data), summarization and combination (i.e., collating many documents into a summary), analysis (i.e., creating summary statistics), and more – but on large archives of unstructured text, imagery, PDFs, and other documents.

LLMs and vision models by themselves help with all of these tasks, but do not wholly solve any of them. Moreover, the lack of abstractions and principles around generative AI makes it very difficult to build a practical, reliable piece of software. Our key thesis is that these **new AI applications need new types of data systems**. We believe it's time to take a step back, rethink the way

we not only build new experiences around data and data systems using LLMs but also address these new emerging areas in a more grounded way by developing new abstractions, principles, best practices, reusable components, and software tools.

Desiderata for such systems include: programmatic access to large collections (millions at least) of arbitrary documents, with the ability to search, extract content from, and transform documents, and, at times, create structured views of them (e.g., that compare patient data or provide summary statistics). Further, these systems need to be relatively efficient, in terms of time and money, to do this in semi-interactive time scales even on very large document collections, and consistent and correct in their answers.

We argue that a declarative interface – inspired by relational languages like SQL – is the answer to these challenges. Specifically, we envision a high level query language the allows users to specify what to extract from their documents, abstracting away from the specific underlying models, prompts, and thresholds. This leaves the query execution system free to choose the specific models and thresholds, in much the way a relational optimizer chooses the best query plan. In such AI systems, however, the optimizer has a much larger variety of execution strategies that it may choose from than in classical query execution systems, with the ability to select different models, synthesize code (using AI models!), tune thresholds, selectively process portions of documents, apply pre-processing and embedding-based indexing, and more. These strategies offer different latency, cost, accuracy, and runtime tradeoffs, with order-of-magnitude variation in these dimensions across them.

This new world of AI-powered data systems brings several exciting research opportunities:

Language. Although a natural choice of language for the database community would be a SQL-like (or perhaps pandas-like) language, we note that natural language interfaces (perhaps chat based) are also declarative (in the "say what I want, not how to do it" sense), and are a natural way for non-expert users like those described above to interface with a data system. We speculate that more people will use NL than SQL in the next 5 years to query data. Still SQL and its ilk will be an important as intermediate languages and targets for code generation.

Multi-objective optimization space. Traditionally, the DB community has focused on one objective: optimize performance. This is changing in the new world and as it becomes a mix of performance, cost, and quality.

Approximate and Adaptive Query Processing. Adaptive techniques are highly relevant in this context as the quality of pipelines depends on the data and complexity of the task, and the quality of the AI agents, and may not be easy to determine a priori.

Fact checking, correctness, and consistency. Users of database systems expect answers to be correct, but new AI systems are unreliable. Here, ideas from human-powered systems [9], such as majority voting and bad-rater detection will be important, but new techniques are needed. These may include asking models to verify each other's answers or check them for bias and toxicity, repeatedly reprocessing inputs to provide estimates of inconsistency or variation, and more.

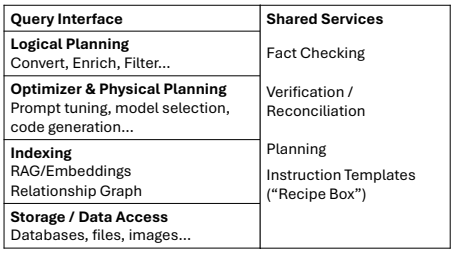

Figure 1: Architecture of an Unbound Database

Need for code generation. Code generation provides an immense opportunity for unstructured data to replace LLM invocations – for example, replacing a call to an LLM with LLM-generated code that achieves a specific data process or extraction task.

In this paper, we describe the key components of such an "unbound database" in more detail, and provide examples of systems we have built to illustrate the potential of these new AI-powered data systems. These include VAAS, a system for querying imagery and video, where a high level declarative specification of video of interest is used to search and extract relevant portions of a video corpus, and Palimpzest, a system for applying high-level declarative operations to arbitrary document collections including PDFs, text, and imagery. We conclude with a set of research challenges that we hope will serve as a roadmap for future research.

2 COMPONENTS OF AN UNBOUND DATABASE

An unbound database, like a conventional database system, consists of several components that together provide an interface to submit queries and compile those queries into an execution plan over documents, images, video, or conventional databases. Many of those components have parallels in conventional data systems, as shown in Figure 1.

These components include: a declarative interface layer where the user specifies their program; a logical planning layer where the system forms a high level operator graph or "plan" for the query; an optimizer; a physical planning layer where implementations such as choice of model, prompt generation, batching and parallelization are done for each logical operator; an indexing layer that provides the ability to lookup documents or images matching particular keywords or criteria, e.g., using embeddings and a vector database; and a a storage layer for accessing documents, images, and other data. In addition there will be a number of shared services that implement features like fact checking and verification or majority voting. We describe each of below, focusing on ways in which these layers differ from their counterparts in conventional database systems and some of the research opportunities in each.

2.1 Query Interface

Much of the excitement around GenAI and databases revolves around the ability to use natural language as a query interface. While we agree that such an approach will be the right one for many application scenarios, we also believe that the inherent ambiguity of natural language (NL) would unnecessarily restrict the set of applications for which unbound database processing could be used if NL was the only interface to the system. That is, if we based our system purely on natural language, applications that

need stricter specificity would not be able to take advantage of the enhanced functionality of an unbound database system.

Thus, we propose a more semantically rigorous interface language for unbound database systems. Such a structured interface language can then be used directly by sophisticated programmers and analysts, or as a target language for compiliers from NL or other user-friendly interfaces.

We believe that a structured declarative interface is a better match for the functionality and range of applications that unbound databases can support. This allows unbound database processing to be treated as an extension of traditional processing and allows us to leverage existing query processing techniques and technology in the development of an unbound query processor. This is the approach we have taken in the architecture described in this section as well as in the development of the prototypes described in Sections 3 and 4.

It is important to note, however, that we see tremendous opportunities for innovation in the end-user interfaces that sit atop the interface language. These include interactive/conversational NL-based interfaces that enable users to iterate and refine their queries as well as visual interfaces such as interactive dashboards.

2.2 Logical Planning / Operators

Traditional relational operators – filters, joins, and aggregates – are still useful and make sense for the unstructured unbound database. What additional ones will be needed? Choosing a small but useful set is a serious design challenge for the data management community. The experience of the data science community and the Pandas package is perhaps a cautionary tale: that package does not have a well-designed set of logical operators, and as a result, performance optimizations have been challenging to implement.

Designing a good operator is surprisingly difficult. It must be high-level enough to permit succinct user-written queries and offer flexibility to the optimizer. At the same time, it must be clear and predictable enough that the user and implementer understand what it will do. For example, natural language is very high-level but in many cases will be too ambiguous for engineering purposes.

Specifically, we propose two operators that encompass a wide range of operations an unbound database will need to perform.

Convert: Structural flexibility is core to the unbound database, and we expect structural data conversions to be ubiquitous. Users will want to convert information from a document to a relational database, or from a data plot to a JSON record. It is easy to imagine applications that involve multiple repeated conversions. Consider a user who wants to retrieve factual claims from a text document, recover data records from a bar chart, or extract relevant portions of a structured database and integrate them into a single output.

Traditional data management might describe these steps as different tasks: textual information extraction, visual extraction, entity matching, schema integration, and so on. But choosing exactly which algorithm is most appropriate for a particular dataset is exactly the kind of low-level decision that the unbound database should be making on the user's behalf. Instead, we think the user should be asked to describe a high-level "convert" task, which the system should implement as best it can.

Convert takes two inputs: (1) a set of data records, described by an accompanying schema (such as WebPage, which might have `html, datecrawled`, and `url` fields) and (2) a desired target schema (such as TextDocument, which has a single `text` field).

The unbound database system then attempts to convert all of the input records into an output set that has the specified target schema. In the above example, the convert operator will strip the HTML to yield a single human-readable text document.

If we want to implement an information extraction task, we might convert a set of TextDocument records into a set of MedicalRecords (which have `appointmentDate, patientName, doctorName`, and `diagnosis` fields). If instead we want to implement a schema integration task, we might simply specify a set of traditional databases as the input, with the target schema as the output. Finally, if we want to perform a data cleaning task, we might specify conversion of a set of ProductListing records (with `productName` and `manufacturer` fields) into a set of CleanedProductListing records (with `twoWordProductName` and `manufacturerStockTicker` fields).

Convert is a high-level operator. For example, it encompasses the `sem_map` and `sem_extract` operations proposed as part of the recent LOTUS project [21]. Further, it is not limited only to text. We can convert an Image to a DogBreed, which entails generating a correct dog breed label for the input image. Or we could convert a TrafficImage into a TrafficDescription record, which might include a `runningRedLight` Boolean field, or a `passengerInCrosswalk` field (we discuss image use cases in more detail in Section 3.)

The user may have to provide additional information that describes the desired cardinalities involved in the conversion. For example, a WebPage that is derived from a conference website might be converted into a single output record of type Proceedings, or into multiple output records of type PaperListing. We have implemented convert as part of the Palimpzest project (Section 4) and it currently has optional parameters for describing desired cardinalities. We note that the output cardinality of convert can also depend on the input (for example, producing on record per paper in a list of proceedings), much like a relational group by.

Enrich: Large AI models do not merely allow us to perform complicated operations on the data in hand. They also allow us to extend existing data with datasets that the user has never even heard of. The enrich operator is intended to be a generalized version of the joinable table search and table union search problems from the dataset discovery literature (e.g., [8, 29]). As with the convert operator, enrich is chosen specifically to permit the system some wide flexibility at implementation time; the user does not need to know which dataset search operator is being used, and in some cases may not even know the corpus over which the search is being executed. The user simply knows that after the enrich step, the dataset gets "bigger and better" in a particular way.

The enrich operator takes a dataset as input and a specification for how to enrich the dataset (in natural language), which might include adding columns or rows, as well as guidance for how much data to add and what types of sources (e.g., the web, model knowledge, etc) to consult.

For example, additional rows can help with assembling a larger training set or test set. In some cases, building a training set might require a couple of steps: use enrich to add additional rows of raw

data objects (such as product web pages), then use convert to extract features and labels. In such cases, the enrich termination criterion might simply count the number of new records.

Or consider using enrich to enable new visualizations. We might want to display real estate listings along two axes: the house price and each house's number of days each year with rainfall. If the starting dataset has house listings, we can use horizontal enrichment to marshal this additional rainfall column.

Building the Recipe Box: Finally, we propose a general method for managing these tradeoffs, which we call a "recipe box". We believe that in many use cases that have semantic ambiguity, agreement amongst operations (from a single user or organization) is of utmost importance. That is, the unbound database should build up a shared set of "recipes" for all of its users, which are employed to resolve ambiguity in user programs – for example, how a company defines complex business terms like "active users", "quarterly revenue", "model performance", and so on. These recipes can be derived from a "quality administrator" or from a training data style process. Recipes might not be shared across installations: the definition of "quarterly earnings" might rightfully differ from company to company. But in many cases, a domain-specific recipe box may be able to deliver both high-level primitives and clear shared semantics for engineering teams.

2.3 Optimizations

The biggest advantage of the declarative approach is that – like SQL – it allows for a range of logical and physical optimizations. Here we list several potential optimizations. In the next sub-section we addresses the challenges of constructing a query plan.

Prompt Tuning: Prompt engineering, the process of asking the LLM the right question, is a very time consuming, trial and error-based process, but can make a significant difference in quality. Maybe more surprisingly, prompt engineering can also significantly impact cost and latency as LLMs service providers often charge extra per input (and output) token and fewer tokens can lead to reduced inference time. Hence a good short prompt, which requests a succinct response, in some cases can lead to significant quality, cost, and latency improvements. Recently, there have been several proposals on how to automatically tune prompts [14, 17], demonstrating that it is possible to tune prompts automatically given a task. We believe that such techniques are not only essential for an unbound database, but that there exists even more potential for prompt optimization given the declarative nature of the task. For example, different strategies could optimize the prompt for cost/latency/quality or certain safe-guards, typically found in prompts, could be disabled if it is known that other operations will perform them later. Similarly, it might be possible to combine several operations into a single prompt to reduce overall cost or to fragment prompts for higher accuracy in order to not confuse the model.

Model Selection: It is an interesting optimization problem to automatically determine what model should be used for each operation in the query pipeline. For example, Anthropic Opus is an extremely powerful – and for some tasks better model than GPT 4o – but also very expensive model. Model selection can include commercial models (e.g., OpenAI, Anthropic), open-source models (e.g., LLaMA) and even up to (on-the-fly) self-trained or fine-tuned model models. For example, for video processing, student/teacher-style training shows significant promise and can yield significant cost/time-savings with little impact on quality [4, 12].

Code Generation: One of the most impactful potential optimizations is to replace parts of the query plan with generated code or other relatively inexpensive operators. For example, consider the legal discovery use case: based on a collection of emails in text format the user wants to extract *date* and *sender name* for all the emails that talk about market domination. Invoking an LLM-model for each individual email could be prohibitively expensive. On the other hand, an LLM can be used to generate a regex expression to extract the date and sender name at almost no cost. Similarly, the LLM could be used to generate a list of keywords related to market domination (e.g., "crush them", "competitor", "leader", "barriers to entry", ...) and create a filter to search emails for those keywords. Such optimizations can reduce latency and cost by orders of magnitude in but also come with their own set of challenges in terms of how to automate their implementation and determine their effectiveness.

Caching: Like most systems, caching can help to significantly improve performance. For unbounded databases we believe all data should be permanently cached (unless the result is invalidated for some reason), as model invocations are so expensive. Moreover, in some cases it might be possible to extract more information in one go over the data for future uses. For example, if the model is trying to determine if a picture contains a car, in doing so the model could provide information about other objects in the picture at little to no extra cost, which might be useful for future queries. However, this "speculative" extraction and caching can also be very expensive in other cases and thus, needs to be carefully optimized.

2.4 Physical Plan Search

There can be orders-of-magnitude difference in cost, quality, and time between a well optimized vs a naive AI plan. As in database systems, we propose a cost-based optimizer to select the best execution plan. However, in contrast to query optimizers the optimization problem is significantly more complex and comes with its own set of unique challenges. Most importantly, there is not a single optimization goal anymore (i.e., time), rather it is a multi-objective optimization, which has to balance cost, time (latency), and quality and it is a research challenge in itself as to how the user specifies the policy. In the current Palimpzest prototype (see Section 4), we allow the user to choose from three policies: cost-, time-, quality-optimized. However, each of these policies only weights one dimension more than the others, but does not "only" optimize for cost, time, or quality. Just picking a single dimension as the goal would only create meaningless results. For example, it is trivial to optimize for cost or time by simply not returning any results (i.e., quality is 0). Clearly, our policy-based approach is just a starting point and much more research is required in providing the users an easy way to express their optimization goals.

Beyond the multi-objective optimization goal, adaptive query processing will play a more crucial role in optimizing the query plan. In contrast to traditional DB optimization, it is very hard to impossible to estimate the quality of certain optimizations just based on statistics. Rather, the system has to try and experiment

on the actual data to determine how successful a certain strategy is, leading to a whole new set of interesting research questions. For example, in the current Palimpzest prototype a mixture of a bandit-based approach and learning from past history is used in order to derive the best execution plan. Interestingly, this approach has many commonalities with AutoML frameworks which often have similar problems e.g., [16, 23].

2.5 Orchestration Operators

As stated above, the use of an operator-based execution framework enables the use of cost-based query optimization to provide efficient access to information contained a wide-variety of data types. Beyond optimization, however, another key advantage of this approach is extensibility in terms of more sophisticated data processing strategies, particularly ones that can orchestrate multiple model calls (either to a particular model or to a variety of models) to enhance accuracy or increase repeatability.

By embedding these steps as operations in a physical query plan, we enable several optimizations: for example, instead of each operator needing to implement its own majority voting, fact checking, or sanitization operations, these operators can be explicitly placed at the right location in the plan.

Example operators that we envision include:

Data Verification: One of the main challenges with using current LLM technology in query processing is the propensity of such models to "hallucinate", that is, to return data that is false or simply made up. Confidence in data extraction and query operators can be enhanced by various strategies, such as asking one model to validate the result obtained from another model, or by asking an LLM to indicate the data sources underlying its answers.

Reconciliation of Multiple Outputs: Another way to enhance accuracy is to send a request to multiple models and to use techniques such as majority voting or EM-based approaches. More sophisticated approaches, such as learning over time that a particular model is good for a particular type of task, are also possible.

"Chain of Thought" Plan Steps: Given a small language for data processing (such as Palimpzest, described in Section 4), LLMs are good at devising programs to perform simple tasks. As such, it should be possible for programmers to write brief descriptions of what they would like to do in English or psuedocode and have the LLM generate a program in the underlying execution language.

Instruction templates: The operator model provides extensibility that enables recipes to be encoded and costed so that they can be appropriately used in a larger query plan to improve cost, accuracy or speed. Further, because of the semantic ambiguity inherent in many AI programs, once a given prompt or program template has been generated, using the "recipe box" design, the database can reuse it in subsequent executions, helping to maintain consistency.

2.6 Indexes

Because arbitrary documents are often quite large, and searching through them can be quite slow, we expect that most data accessed by an unbound database system will first be pre-processed and indexed in some form to support efficient access.

For text, imagery, and video, embeddings, where we transform a high dimensional object (say a paragraph of text or a single image) into some lower dimensional space (say, a 512 element vector), are a natural fit. Recent years have shown that transformer based models [26] work particularly well for embeddings. For example, BERT [7] and its many variants are often used for text embeddings (although in recent years BERT based model performance has been eclipsed by new representations.) For imagery, again there many possible choices, but of particular interest for query systems are *cross-modal* embeddings where text and imagery are encoded into the same space, such that an image and a textual description of the image will have approximately the same embedding [22].

Given a corpus of data that has been embedded in some way, a common next step is to store the data in an index to facilitate efficient retrieval of related documents. A typical setup is to encode every document using an embedding (or to divide documents in sections and embed each section) and store these embeddings in a vector database, which provides an approximate nearest neighbor (ANN) query facility. Then, given a query vector (e.g., of a document embedding using the same encoder as the documents stored in the database), an ANN query can be issued to find related documents. Despite the popularity of this approach, there are several interesting questions for database researchers, including:

Attribute-based retrieval. Performing retrieval both based on attribute values as well as ANN vector search will often be required. Although some existing vector stores support this use case, they typically do so either by first performing a vector search and then constraining on attribute values or search on attribute values and then scanning through the retrieved vectors. Other approaches – e.g., creating clusters of documents with similar attribute values that are indexed by ANN techniques – may offer significant performance gains.

Choice of embeddings. There are often many possible embeddings for the same object, and different embeddings may be better for particular use cases (i.e., retrieval vs. classification.) Further the choice of how to divide documents (images or text) into chunks, whether the chunks should overlap and by how much, etc. has a significant impact on retrieval of both text and images.

Embedding alternatives. Although embeddings are an effective general purpose tool for summarizing documents, many other higher-level representations are possible. For example, much of the work querying over images and video encodes images using a higher-level semantic representation, i.e., encoding images as a list of the objects that are in them [4, 12, 13]. Similar approaches can be applied to text – i.e., summarizing before embedding. It is unclear how storage and retrieval performance (both in terms of time and accuracy) vary between these domain-specific methods and general purpose RAG methods.

2.7 Storage / Data Access

Below the indexing layer, an unbound database will clearly need access to a variety of data types, including tabular databases as well as archives of text, image, video, audio, and more. In a retrieval-only system, these may simply be immutable collections accessed via an index or an exhaustive scan.

More interesting questions arise when updates and insertions are considered. A selling point of database systems has long been that they provide strong consistency in the face of changing data, but it's unclear what consistency even means in an AI world where many questions have ambiguous or multifarious answers.

Some basic principles are straightforward: a system might provide a transaction-like guarantee that the set of documents considered throughout the transaction would not change, or the system might provide a guarantee that an answer given earlier in the transaction might be the same for a later invocation.

Many other aspects of updates are much less clear. For example, is it possible to issue statements that modify the contents of documents? What are the syntax and semantics of such modifications? Given that a common operation in these systems is likely to produce derived data, e.g., summaries or extractions from base data, how is such derived data tracked, and how are relationships to base data preserved? What if base data changes or is removed? Similar issues arise in data provenance, versioning, and materialized views for relational systems, although the details are likely quite different.

Building on these foundational components we have built several new AI Data Systems which will help more directly with particular end-to-end tasks or improve the interface of existing systems. These include Video Querying (Section 3) and a new AI-powered data science system called Palimpzest (Section 4).

3 VIDEO QUERYING

In this section we describe VAAS (Video Analytics at Scale) [25], a system built for declarative video querying that is a specific embodiment of the architecture described above. It processes stored video using a high level declarative program (specified graphically) that is optimized into an efficient physical plan that attempts to minimize processing time while maximizing result quality. As with LLMs, recent advances in image and video processing algorithms that can identify objects, segment images, and produce textual descriptions of scenes mean that it is possible to build systems that "understand" images in a transformative way.

The motivation for such systems is that there are many settings where people have large archives of video and they want to answer questions over them or search through them to find events of interest. Examples include:

Autonomous driving. Autonomous driving requires millions of hours of video to train and and improving models requires identifying challenging scenarios from video – such as bikes in the snow or vehicles driving the wrong way – to evaluate algorithms on.

Traffic cameras. A city might want to analyze intersection cameras, of which they may have hundreds or thousands, to find places where people engage in risky behavior – e.g., running red lights – to determine where to implement traffic calming measures.

Biology. A biologist may want to analyze wildlife camera images to find specific animals or behaviors, such as eating or mating.

Although deep learning models, such as image segmentation and object tracking, can be used to implement many of these examples, a naive application of such models would be prohibitively expensive. For example, on a high-end GPU on Amazon AWS, where a state-of-the-art object tracker like Mask-RCNN [11] runs at about 30 frames per second, processing one month of video from 100 cameras would cost about $72,000! Put differently, a single $80 camera can capture more than 946 million frames a year.

Clearly, a naive search strategy that applies expensive AI models to every frame would require a huge and likely infeasible amount of computation, even for modest collections. Also, video search is likely to be used in interactive settings when fast results are especially valuable to the user. For both economic and user experience reasons, a video search component should be as fast as possible, even in the face of huge data collections.

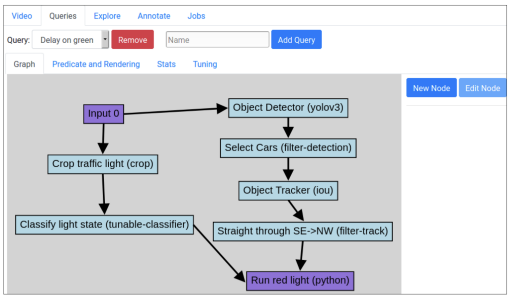

Figure 2: Example visual query specification for a redlight running query in VAAS.

Fortunately, there is a potential opportunity in video search: the vast majority of video frames are irrelevant for most queries, so only a small fraction of model invocations will yield valuable results. If we could somehow identify the potentially-valuable frames without invoking the visual model, we could sidestep all the fruitless frames and save a huge amount of runtime and computation.

Queries in VAAS are specified through a largely declarative visual querying interface (see Figure 2). They do not specify a specific execution plan, but instead describe logical operators to apply to the data. They thereby enable potential system optimizations that can avoid unnecessary work, such as re-ordering operators or consuming video out-of-order, exploiting novel semantic properties of video operators. For example, in the query shown in Figure 2, each detection needs to both contain a red light and a car passing through a specific sequence of rectangles (the "straight through" box). If red lights are less common than cars passing through the intersection, the best query plan is likely to first detect red lights, then look for tracks of vehicles.

VAAS's most common operations are to filter and aggregate video after using convert operators to cast raw images into specific types of object detections and tracks (e.g., vehicles and paths). Filters include detecting one or more objects (in a bounding box) or filtering the path of a trajectory through a bounding box (possibly defined by another object or trajectory). Queries can also apply temporal predicates, e.g., tracks that pass through box A then box B (as opposed to just A and B).

Optimization Techniques. Because VAAS provides a high level interface for expressing queries over video, there are a variety of opportunities for automatic optimization of queries that we plan to pursue to reduce the number or resolution of video frames that need to be processed, or the sophistication of models that need to be run to produce accurate query results. Specific optimizations the we have implemented include reducing the video frame rate adaptively, filtering out partial detections (tracks) that are unlikely to satisfy the query, focusing on portions of the viewframe of interest, and applying declarative ordering optimizations to query predicates. These optimizations are described in several recent papers published in SIGMOD and VLDB [3, 4].

Storage and Indexing. Of course, operating directly on massive archives of video will be quite slow. For this reason, we (and several others) have developed indexing techniques that pre-detect objects and paths (tracks) of those objects in video [4]. These pre-detected objects can then be embedded and stored in a vector database so support efficient retrieval of objects of interest [13].

4 PALIMPZEST

In this section, we describe a second embodiment of an unbound database we have built, called Palimpzest [18]. Palimpzest is a programming tool that makes it possible to query text and images just like a relational database can query tables. It enables engineers to write succinct, declarative code that can be compiled into optimized programs. It is targeted at a variety of workloads, including large-scale information extraction, data integration, data discovery from medical and scientific literature, image understanding tasks, and multimodal analytics. Palimpzest is declarative in that users write a high level program, and it considers a range of logical and physical optimizations, yielding a set of possible concrete executable programs. Palimpzest then estimates the cost, time, and quality of each one, then chooses a program based on runtime user preferences. The system is designed to be extensible, so that new optimizations can be easily added in the future. Just as relational databases allow users to write SQL queries more quickly and correctly than they could by writing traditional code, Palimpzest will allow engineers to write better AI programs more quickly than they could unaided. It uses many of the components described above, including prompt tuning, model selection, cost modeling, and guard rails to generate optimized implementations of declarative programs.

A core challenge in building Palimpzest is creating an optimizer that can marshal many optimizations to meet a user's cost, runtime, and quality goals. By using a language that is high-level, type-focused, and declarative — rather than the low-level prompting and coding method pursued by naive programming and some other frameworks (e.g., LangChain)— we believe Palimpzest can exploit many optimizations that would not otherwise be available. Another key challenge involves designing a programming interface which simultaneously enables engineers to express the broadest possible set of AI programs, while imposing structure on their programs that the optimizer can exploit. To this end, we created a Python library which implements a thin abstraction over an underlying relational algebra. The core intellectual operation in Palimpzest is the relational convert operator, which transforms an object of one user-defined schema to another. This operator — which is implemented using a variety of methods, often based on foundation models — allows the programmer to implement many AI tasks in a relational and optimizable style.

To illustrate Palimpzest, consider the declarative program shown in Figure 3; this program iterates over a set of emails to identify those that refer to a potentially fraudulent scheme (this is a workload derived from the Enron emails):

In this short example program, the user wants to identify emails that are not quoting from sources outside of Enron and that reference fraudulent investment vehicles. As a first step, the programmer uses Palimpzest to create a custom schema for the input dataset of Emails – on lines 3-6. In this case, Email is a subclass of TextFile,

```
1  import palimpzest as pz
2
3  class Email(pz.TextFile):
4    """Represents an email, subclass of 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)
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 and execute plan
14 policy = pz.MinimizeCostAtFixedQuality(min_quality=0.8)
15 results = pz.Execute(emails, policy=policy)
```

Figure 3: Palimpzest code for the Legal Discovery workload.

which is defined in Palimpzest's core library and inherits directly from the base Schema class. Starting on line 9, the user begins to describe data processing actions, beginning with instantiating an initial Dataset that adheres to the Email schema. The source string "enron-emails" uniquely identifies a set of files that have been pre-registered with Palimpzest. The code on line 9 transforms the raw input data objects into the Email Schema and stores the results in the emails Dataset. On line 10, the program filters emails for the subset which are not quoting from news articles. On line 11 the program filters for emails which discuss fraudulent investments.

The programmer takes two more steps: on line 14, they specify a policy that describes how the system should choose among multiple possible implementations of the steps described so far. (In this case, the plan with the lowest expected financial cost, subject to a lower bound on quality, is preferred.) Finally, on line 15, the programmer asks Palimpzest to Execute() the program; this entails generating a logical execution plan, generating multiple optimized physical execution plan candidates, choosing one according to the specified policy, and then executing the code and yielding results. Programs written with Palimpzest are executed lazily, thus no actual data processing occurs until line 15.

The user-provided description strings for the schema fields and filters comprise both a way for the developer to specify correct program output, and a way for the system to find a high-quality implementation. These strings are provided to underlying (often Generative AI-based) operators which use them when constructing internal prompts. In contrast to prompt engineering, we do not intend for users to expend significant effort tuning these descriptions. Instead, Palimpzest tunes the prompts automatically for the user.

As noted above, the key component of Palimpzest is an optimization framework that automatically decides how to best execute the program. When running an input user program, Palimpzest considers a range of logical and physical optimizations, then yields a set of possible concrete executable programs. Palimpzest estimates the cost, time, and quality of each one, then chooses a program based on runtime user preferences. The system is designed to be extensible, so that new optimizations can be easily added in the future. Just as relational databases allow users to write database queries more quickly and correctly than they could by writing traditional code, Palimpzest allows engineers to write better AI

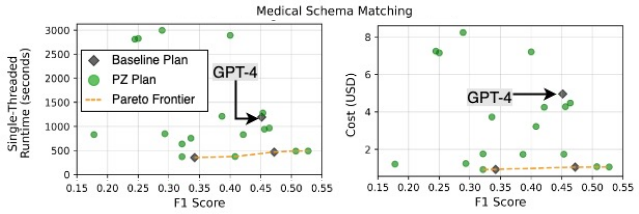

Figure 4: Runtime, cost, and quality of plans found by the declarative Palimpzest system.

programs more quickly than they could unaided. Examples of optimizations Palimpzest considers, driven by the components we are developing in the overall New AI Data Platform, include:

Using the Prompt Tuning module, Palimpzest will optimize wording and decide on a general prompting strategy (e.g., zero-shot, few-shot, chain-of-thought, ReAct, etc.). Using the Model Selection module, Palimpzest will try to pick the best model for each subtask in the program, balancing time, cost, and quality; for example for easy tasks it might run a small model on a CPU, and for more difficult tasks it might call out to GPT-4o. With the Code Synthesis and Quality Estimator modules, Palimpzest will decide whether each subtask is best implemented by a foundation model query, synthesized code, or a locally-trained student model. Furthermore, it will consider how to combine tasks to improve GPU cache utilization, and how to avoid running over LLM context limits. All Palimpzest outputs will be validated using the Fact Checking module.

Finally, when scaling out to a larger dataset, Palimpzest faces additional challenges in selecting an efficient execution plan. Even if the system performs well on a small dataset, it may require redesign to ensure reasonable runtime, cost, and performance at a larger scale. This may involve enabling parallelism for each component and scheduling these parallelized components on multiple GPUs and CPUs for optimal efficiency.

Figure 4 shows the results of running the Palimpzest system on a schema matching task (extracting cancer data from several PDFs and creating a merged table). Each of the green dots in the figure represents one candidate plan generated by Palimpzest for this task, with a plan that uses GPT4 for every step in the program shown with an arrow. The left plot shows plan quality on the X axis (measured as F1 score relative to a human-generated ground truth result) and the Y axis shows runtime in seconds. On the right, F1 score is plotted against dollar cost. For this particular program, Palimpzest generates plans that are 20% of the cost of GPT4 while achieving significantly higher F1 scores at lower runtimes.

5 RELATED WORK

We note that there is currently a vast and ever growing literature in the AI community on LLMs and vision models. Rather than trying to cover all of this work, we focus here on work from the database community that is most relevant to our proposal.

There have been a other recent attempts to mix LLMs with the relational query model, and the amount of work in this area has been growing rapidly [6, 18, 21, 24]. These systems differ along several design dimensions: the use of natural language at query-writing time, the semantics of operators, and the optimizations supported. Castro Fernandez, *et al.* presented a high-level vision of possible changes in data management due to recent AI advances [10].

A key technical ability of the unbound database will be to substitute cost- and quality-effective implementations for expensive AI operations. Notable efforts in the data management community include model cascades, especially for vision tasks [1, 12, 15]; and code synthesis methods [2, 5].

There is growing recognition in other communities about the need to combine software systems concepts and AI elements. SGLang [28] is an early example of a tool designed to build "compound AI systems" [27]. The vision sketched in this paper can be thought of as a compound AI system tailored for data management tasks.

Finally, we note that many of the challenges around the unbound database were considered roughly a decade ago in the context of crowd-powered databases. These systems [9, 19, 20] similarly had new declarative operators that were not implemented through conventional computation; instead of processing tasks with an expensive AI model, they used human answers. Although many technical details have changed, we have found it instructive to review and reconsider these projects.

6 CONCLUSION

We presented our vision of an unbound database, which provides a declarative interface to all of the world's data, spanning not just tables but text, images, video, documents and more. While new AI models are a necessary element, such a system will need many other novel components to deliver scale and robustness needed for many enterprise-class applications. These include: new interfaces, which may be be based on natural language or new programming languages; optimizations like caching, prompt tuning, batching, and code generation to avoid expensive LLM calls on every on every input element; cost/quality/performance optimizers that help users navigate tradeoffs between runtime and dollars; techniques to ensure repeatability and verifiability of answers; and more. We presented two examples of such systems, VAAS and Palimpzest, and illustrated how they make use of declarative optimization techniques to explore performance tradeoffs.

SPEAKER BIOGRAPHIES

Samuel Madden is a the College of Computing Distinguished Professor of Computing at MIT. His research interests include databases, distributed computing, and networking. Research projects include learned database systems, the C-Store column-oriented database system, and the CarTel mobile sensor network system. Madden heads the Data Systems Group at MIT and the Data Science and AI Lab (DSAIL), an industry supported collaboration focused on developing systems that use AI and machine learning.

Madden was named one of Technology Review's Top 35 Under 35 in 2005 and an ACM Fellow in 2020, and is the recipient of several awards, including an NSF CAREER award, a Sloan Foundation Fellowship, the ACM SIGMOD Edgar F. Codd Innovations Award, and "test of time" awards from VLDB, SIGMOD, SIGMOBILE, and SenSys. He is the co-founder and Chief Scientist at Cambridge Mobile Telematics, which develops technology to make roads safer and drivers better.

REFERENCES

[1] Michael R Anderson, Michael Cafarella, German Ros, and Thomas F Wenisch. 2019. Physical representation-based predicate optimization for a visual analytics database. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1466–1477.

[2] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv preprint arXiv:2304.09433* (2023).

[3] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael J. Cafarella, Tim Kraska, and Sam Madden. 2020. MIRIS: Fast Object Track Queries in Video. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1907–1921. https://doi.org/10.1145/3318464.3389692

[4] Favyen Bastani and Samuel Madden. 2022. OTIF: Efficient Tracker Pre-processing over Large Video Datasets. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 2091–2104. https://doi.org/10.1145/3514221.3517835

[5] Zui Chen, Lei Cao, Sam Madden, Ju Fan, Nan Tang, Zihui Gu, Zeyuan Shang, Chunwei Liu, Michael Cafarella, and Tim Kraska. 2023. Seed: Simple, efficient, and effective data management via large language models. *arXiv preprint arXiv:2310.00749* (2023).

[6] Hanjun Dai, Bethany Yixin Wang, Xingchen Wan, Bo Dai, Sherry Yang, Azade Nova, Pengcheng Yin, Phitchaya Mangpo Phothilimthana, Charles Sutton, and Dale Schuurmans. 2024. UQE: A Query Engine for Unstructured Databases. arXiv:2407.09522 [cs.DB] https://arxiv.org/abs/2407.09522

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[8] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *Proceedings of the VLDB Endowment* 16, 7 (2023), 1726–1739. https://doi.org/10.14778/3587136.3587146

[9] Amber Feng, Michael J. Franklin, Donald Kossmann, Tim Kraska, Samuel Madden, Sukriti Ramesh, Andrew Wang, and Reynold Xin. 2011. CrowdDB: Query Processing with the VLDB Crowd. *Proc. VLDB Endow.* 4, 12 (2011), 1387–1390. http://www.vldb.org/pvldb/vol4/p1387-feng.pdf

[10] Raul Castro Fernandez, Aaron J. Elmore, Michael J. Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How Large Language Models Will Disrupt Data Management. *Proc. VLDB Endow.* 16, 11 (2023), 3302–3309. https://doi.org/10.14778/3611479.3611527

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. 2017. Mask R-CNN. *CoRR* abs/1703.06870 (2017). arXiv:1703.06870 http://arxiv.org/abs/1703.06870

[12] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: optimizing neural network queries over video at scale. *Proc. VLDB Endow.* 10, 11 (aug 2017), 1586–1597. https://doi.org/10.14778/3137628.3137664

[13] Daniel Kang, John Guibas, Peter D. Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2022. TASTI: Semantic Indexes for Machine Learning-based Queries over Unstructured Data. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1934–1947. https://doi.org/10.1145/3514221.3517897

[14] 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).

[15] Ferdi Kossmann, Ziniu Wu, Alex Turk, Nesime Tatbul, Lei Cao, and Samuel Madden. 2024. CascadeServe: Unlocking Model Cascades for Inference Serving. *CoRR* abs/2406.14424 (2024). https://doi.org/10.48550/ARXIV.2406.14424 arXiv:2406.14424

[16] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research* 18 (2018), 1–52.

[17] Xiang Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. 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)*. Association for Computational Linguistics, 4582–4597.

[18] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. arXiv:2405.14696 [cs.CL] https://arxiv.org/abs/2405.14696

[19] Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. 2011. Crowdsourced Databases: Query Processing with People. In *Fifth Biennial Conference on Innovative Data Systems Research, CIDR 2011, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*. www.cidrdb.org, 211–214. http://cidrdb.org/cidr2011/Papers/CIDR11_Paper29.pdf

[20] Hyunjung Park, Richard Pang, Aditya G. Parameswaran, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. 2012. Deco: A System for Declarative Crowdsourcing. *Proc. VLDB Endow.* 5, 12 (2012), 1990–1993. https://doi.org/10.14778/2367502.2367555

[21] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. LOTUS: Enabling Semantic Queries with LLMs Over Tables of Unstructured and Structured Data. arXiv:2407.11418 [cs.DB] https://arxiv.org/abs/2407.11418

[22] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. *CoRR* abs/2103.00020 (2021). arXiv:2103.00020 https://arxiv.org/abs/2103.00020

[23] Zeyuan Shang, Emanuel Zgraggen, and Tim Kraska. 2019. Alpine Meadow: A System for Interactive AutoML. In *Proceedings of the MLSys: Workshop on Systems for ML at NeurIPS*.

[24] Matthias Urban and Carsten Binnig. 2023. CAESURA: Language Models as Multi-Modal Query Planners. *arXiv preprint arXiv:2308.03424* (2023).

[25] vaas [n. d.]. https://vaas.csail.mit.edu/docs/introduction.html.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 http://arxiv.org/abs/1706.03762

[27] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. 2024. The Shift from Models to Compound AI Systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/.

[28] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2023. Efficiently programming large language models using sglang. *arXiv preprint arXiv:2312.07104* (2023).

[29] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)* (Amsterdam, Netherlands). ACM, 847–864. https://doi.org/10.1145/3299869.3319901