Relational Programming with Foundation Models

Ziyang Li, Jiani Huang, Jason Liu, Felix Zhu, Eric Zhao, William Dodds, Neelay Velingker, Rajeev Alur, Mayur Naik
University of Pennsylvania
liby99@seas.upenn.edu, jianih@seas.upenn.edu, jasonhl@seas.upenn.edu, zhufelix@seas.upenn.edu, zhaer@seas.upenn.edu, wdodds@sas.upenn.edu, neelay@seas.upenn.edu, alur@seas.upenn.edu, mhnaik@seas.upenn.edu

Abstract
Foundation models have vast potential to enable diverse AI applications. The powerful yet incomplete nature of these models has spurred a wide range of mechanisms to augment them with capabilities such as in-context learning, information retrieval, and code interpreting. We propose VIEIRA, a declarative framework that unifies these mechanisms in a general solution for programming with foundation models. VIEIRA follows a probabilistic relational paradigm and treats foundation models as stateless functions with relational inputs and outputs. It supports neuro-symbolic applications by enabling the seamless combination of such models with logic programs, as well as complex, multi-modal applications by streamlining the composition of diverse sub-models. We implement VIEIRA by extending the SCALLOP compiler with a foreign interface that supports foundation models as plugins. We implement plugins for 12 foundation models including GPT, CLIP, and SAM. We evaluate VIEIRA on 9 challenging tasks that span language, vision, and structured and vector databases. Our evaluation shows that programs in VIEIRA are concise, can incorporate modern foundation models, and have comparable or better accuracy than competitive baselines.

Introduction
Foundation models are deep neural models that are trained on a very large corpus of data and can be adapted to a wide range of downstream tasks (Bommasani et al. 2021). Examples of foundation models include language models (LMs) like GPT (Bubeck et al. 2023), vision models like Segment Anything (Kirillov et al. 2023), and multi-modal models like CLIP (Radford et al. 2021). While foundation models are a fundamental building block, they are inadequate for programming AI applications end-to-end. For example, LMs hallucinate and produce nonfactual claims or incorrect reasoning chains (McKenna et al. 2023). Furthermore, they lack the ability to reliably incorporate structured data, which is the dominant form of data in modern databases. Finally, composing different data modalities in custom or complex patterns remains an open problem, despite the advent of multi-modal foundation models such as VILT (Radford et al. 2021) for visual question answering.

Various mechanisms have been proposed to augment foundation models to overcome these limitations. For example, PAL (Gao et al. 2023), WebGPT (Nakano et al. 2021), and Toolformer (Schick et al. 2023) connect LMs with search engines and external tools, expanding their information retrieval and structural reasoning capabilities. LMQL (Beurer-Kellner, Fischer, and Vechev 2022) generalizes pure text prompting in LMs to incorporate scripting. In the domain of computer vision (CV), neuro-symbolic visual reasoning frameworks such as VisPROG (Gupta and Kembhavi 2022) compose diverse vision models with LMs and image processing subroutines. Despite these advances, programmers lack a general solution that systematically incorporates these methods into a single unified framework.

In this paper, we propose VIEIRA, a declarative framework for programming with foundation models. VIEIRA follows a probabilistic relational paradigm due to its theoretical and practical versatility. Structured data is commonly stored in relational databases. Relations can also represent structures such as scene graphs in vision and abstract syntax trees in natural and formal languages. Moreover, extensions...
for probabilistic and differentiable reasoning enable the integration of relational programming with deep learning in neuro-symbolic frameworks like DeepProbLog (Manhaeve et al. 2018) and SCALLOP (Li, Huang, and Naik 2023).

In Vieira, relations form the abstraction layer for interacting with foundation models. Our key insight is that foundation models are stateless functions with relational inputs and outputs. Fig. 1a shows a Vieira program which invokes GPT to extract the height of mountains whose names are specified in a structured table. Likewise, the program in Fig. 1b uses the image-text alignment model CLIP to classify images into discrete labels such as cat and dog. Fig. 1c shows relational input-output examples for the two programs. Notice that the CLIP model also outputs probabilities that allow for probabilistic reasoning.

We implement Vieira by extending the SCALLOP compiler with a foreign interface that supports foundation models as plugins. We implement a customizable and extensible plugin library comprising 12 foundation models including GPT, CLIP, and SAM. The resulting unified interface enables a wide spectrum of applications with benefits such as reduced hallucination, retrieval augmentation, and multimodal compositionality. We evaluate Vieira on 9 applications that span natural language reasoning, information retrieval, visual question answering, image generation, and image editing. For these applications, we explore diverse methods for programming with foundation models, such as neuro-symbolic reasoning, combining semantic searching with question answering, and modularly composing foundation models. We not only observe on-par or superior performance of our solutions compared to competitive baselines, but also demonstrate their succinctness and ease-of-use.

We summarize our contributions as follows: (1) we introduce a new approach based on relational programming to build applications on top of foundation models; (2) we implement an extensible plugin library of 12 programmable foundation models; and (3) we evaluate Vieira on 9 benchmark tasks, and demonstrate comparable or better no-training accuracy than neural-only as well as task-specific baselines. Our framework, plugin library, and evaluations are open-source and available at https://github.com/scallop-lang/scallop.

Related Work

Neuro-symbolic methods. These methods combine the complementary benefits of neural learning and symbolic reasoning. They include domain-specific solutions (Yi et al. 2018; Mao et al. 2019; Li et al. 2020; Wang et al. 2019; Xu et al. 2022; Chen et al. 2020; Minervini et al. 2020) as well as general programming frameworks, such as DeepProbLog (Manhaeve et al. 2018) and SCALLOP (Li, Huang, and Naik 2023). These methods typically concern training or fine-tuning neural models in the presence of logical programs, whereas we target building applications atop foundation models with zero-shot or few-shot examples. Another recent work, the STAR framework (Rajasekharan et al. 2023) also connects a language model (neural) to an answer set programming reasoner (symbolic). It is conceptually similar to Vieira but only focuses on natural language understanding and does not support probabilistic reasoning.

Foundation models. These models target different modalities and domains (Touvron et al. 2023; OpenAI 2023; Radford et al. 2021; Kirillov et al. 2023; Radford et al. 2021). Their reasoning capabilities continue to improve with larger context sizes (Ratner et al. 2023), smarter data selection (Adadi 2021), and the discovery of new prompting methods, such as chain-of-thought (Wei et al. 2023; Kojima et al. 2022), self-consistency (Wang et al. 2023), and ReAct (Yao et al. 2023). Vieira is orthogonal to these techniques and stands to further enhance the robustness and reliability of foundation models in end-to-end AI applications.

Tools aiding language models. There are many efforts that seek to improve the reasoning abilities of language models (LMs) by incorporating external programs and tools (Gao et al. 2023; Schick et al. 2023; Nakano et al. 2021; Davis and Aaronson 2023). For instance, AutoGPT (Richards 2023) and TaskMatrix.AI (Liang et al. 2023) allows black-box LMs to control symbolic reasoning by invoking commands or calling APIs. On the other hand, many works attempt to extract structured information from LMs for downstream tasks (Gupta and Kembhavi 2022; Beurer-Kellner, Fischer, and Vechev 2022). Vieira unifies these two strategies for augmenting model capabilities, and extends them into a glue language for composing multi-modal foundation models.

Language

Vieira employs a declarative logic programming language based on Datalog (Abiteboul, Hull, and Vianu 1994). In this section, we present the core language and its foreign interface for incorporating diverse foundation models.

Core Language

Relations and data types. The fundamental data type in Vieira is set-valued relations comprising tuples of statically-typed primitive values. Besides the standard primitive types such as integers (e.g. 32) and string (String), Vieira introduces two additional types for seamless integration of foundation models: Tensor and Algebraic Data Types (ADTs). For example, we can declare a relation named image to store tuples of image IDs and image Tensors:

```plaintext
type image(id: i32, img: Tensor)
```

The contents of this relation can be specified via a set of tuples using the built-in foreign function $load_image:

```plaintext
rel Image = {((0, $load_image("cat.png")), ...)
```

ADTs in Vieira enable the specification of domain specific languages (DSLs) to bridge structured and unstructured data. For example, the following DSL for visual question answering (VQA) describes queries to retrieve scene objects, count objects, and check the existence of objects:

```plaintext
type Query = Scene | Filter(Query, String) | Count(Query) | Exists(Query) | ...;
const MY_QUERY = Count(Filter(Scene(), "ball"))
```

10636
Logical reasoning. Being based on Datalog, VIEIRA supports defining Horn rules, thereby allowing logical reasoning constructs such as conjunction, disjunction, recursion, stratified negation, and aggregation. Recursion is particularly useful for inductively defining the semantics of a DSL. For example, a (partial) semantics for the above DSL is defined as follows, where eval_o and eval_n are recursively defined to evaluate objects and numbers, respectively:

```python
# Each result is tagged by a probability
probs = clip_model(img, labels)
yield (prob, (label,)) # prob::(label,)
```

Figure 2: Snippet of Python implementation of the foreign attribute clip which uses the CLIP model for image classification. Notice that the FA clip returns the FP run_clip argument (using free or omitted for brevity). Semantically, FPs are functions that take in a tuple of bounded arguments and return a list of tuples of free arguments. The runtime of VIEIRA performs memoization on FP results to avoid redundant computation. Optionally, FPs can tag a probability to each returned tuple for further probabilistic reasoning.

Foreign Attribute (FA). In VIEIRA, attributes can be used to decorate declarations of predicates. They are higher-order functions that take in the provided arguments and the decorated predicate to return a new predicate. The syntax for using an attribute to decorate a predicate is:

```python
@ATTR(POS_ARG, ..., KEY=KW_ARG, ...) type PRED([bound|free]? ARG: TYPE, ...)
```

The attribute is applied prior to the compilation of VIEIRA programs. For interfacing with foundation models, the positional and keyword arguments are particularly helpful in configuring the underlying model, hiding low-level details. Fig. 2 illustrates one succinct implementation of the FA that enables the use of the CLIP model shown in Fig. 1b.

Foundation Models

VIEIRA provides an extensible plugin framework that adapts to the evolving landscape of foundation models. In this work, we have implemented 7 plugins, covering 12 foundation models, all through the foreign interface. Our design principle for the interface is three-fold: simplicity, configurability, and compositionality. In this section, we present several representative predicates and attributes which substantially support the applicability of VIEIRA to diverse machine learning tasks.

Text completion. In VIEIRA, language models like GPT (OpenAI 2023) and LLaMA (Touvron et al. 2023) can be used as basic foreign predicates for text completion:

```python
extern type gpt(bound p: String, a: String)
rel ans(a) = gpt("population of NY is", a)
```

In this case, gpt is an arity-2 FP that takes in a String as the prompt and produces a String as the response. It uses the model gpt-3.5-turbo by default. To make the interface more relational and structural, we provide an FA:
Here, we declare a relation named population which produces a population number (num) given a location (loc) as input. Notice that structured few-shot examples are provided through the argument examples.

**Semantic parsing.** One can directly configure language models to perform semantic parsing. For instance, the semantic parser for the simple query DSL (partially defined in the Language section) can be declared as follows:

```plaintext
@gpt("the population of {{loc}} is {{num}}", examples=[{"NY", 8468000}, {"...}] type population(bound loc: String, num: u32)
```

Internally, the language model is expected to generate a fully structured query in its string form. Then, Vieira attempts to parse the string to construct actual ADT values. In practice, the success of semantic parsing depends heavily on the design of the DSL, involving factors like intuitiveness (e.g., names and arguments of ADT variants) and complexity (e.g., number of possible ADT variants).

**Relational data extraction.** Structural relational knowledge available in free-form textual data can be extracted by language models. We introduce a foreign attribute @gpt_extract_relation for this purpose. For instance, the following declared predicate takes in a context and produces (subject, object, relation) triplets:

```plaintext
@gpt_extract_relation("Extract the implied kinship relations", examples=["Alice and her son Bob went to...", ["Alice", "Bob", "son", ...]])
```

This attribute differs from the text completion attribute in that it can extract an arbitrary number of facts. The underlying implementation prompts LMs to respond with JSON-formatted strings, allowing structured facts to be parsed.

**Language models for textual embedding.** Textual embeddings are useful in performing tasks such as information retrieval. The following example declares an FP encapsulating a cross-encoder (Nogueira and Cho 2019):

```plaintext
@cross_encoder("nli-deberta-v3-xsmall")
```

In the last line, we compute the cosine-similarity of the encoded embeddings using a soft-join on the variable e. As a result, we obtain a probabilistic fact like $0.9::\text{sim}(e)$ whose probability encodes the cosine-similarity between the textual embeddings of "cat" and "neko".

**Image classification models.** Image-text alignment models, such as CLIP (Radford et al. 2021), can naturally be used as zero-shot image classification models. Fig. 1b shows an example usage of the @clip attribute. We also note that dynamically-generated classification labels can be provided to CLIP via a bounded argument in the predicate.

**Image segmentation models.** OWL-ViT (Minderer et al. 2022), Segment Anything Model (SAM) (Kirillov et al. 2023), and DSFD (Li et al. 2018) are included in Vieira as image segmentation (IS) and object localization (LOC) models. IS and LOC models can provide many outputs, such as bounding boxes, classified labels, masks, and cropped images. For instance, the OWL-ViT model can be used and configured as follows:

```plaintext
@owl_vit("human face", "rocket")
```

Here, the find_obj predicate takes in an image, and finds image segments containing "human face" or "rocket". According to the names of the arguments, the model extracts 3 values per segment: ID, label, and cropped image. Note that each produced fact will be associated with a probability, representing the confidence from the model.

**Image generation models.** Visual generative models such as Stable Diffusion (Rombach et al. 2022) and DALL-E (Ramesh et al. 2021) can be regarded as relations as well. The following example shows the declaration of the gen_image predicate, which encapsulates a diffusion model:

```plaintext
@stable_diffusion("stable-diffusion-v1-4")
```

As can be seen from the signature, it takes in a String text as input and produces a Tensor image as output. Optional arguments such as the desired image resolution and the number of inference steps can be supplied to dictate the granularity of the generated image.

**Tasks and Solutions**

We apply Vieira to solve 9 benchmark tasks depicted in Fig. 3. Table 1 summarizes the datasets, evaluation metrics, and the foundation models used in our solutions. We elaborate upon the evaluation settings and our solutions below.

**Date reasoning (DR).** In this task adapted from BIG-bench (Srivastava et al. 2023), the model is given a context and asked to compute a date. The questions test the model’s temporal and numerical reasoning skills, as well as its grasp of common knowledge. Unlike BIG-bench where multiple-choice answers are given, we require the model to directly produce its answer in MM/DD/YYYY form.

Our solution leverages GPT-4 (5-shot) for extracting 3 relations: mentioned dates, duration between date labels, and the target date label. From here, our relational program iterates through durations to compute dates for all date labels. Lastly, the date of the target label is returned as the output.

**Tracking shuffled objects (TSO).** In this task from BIG-bench, a textual description of pairwise object swaps among people is given, and the model needs to track and derive which object is in a specified person’s possession at the end.

---

Footnote: In this work, $k$ in "$k$-shot" means the number of examples provided to the LM component within the full solution. Each example is a ground-truth input-output pair for the LM.
Figure 3: Benchmark tasks. The top of each box lists the dataset(s) and the foundation models used in our solutions.

There are three difficulty levels depending on the number of objects to track, denoted by \( n \in \{3, 5, 7\} \).

Our solution for tracking shuffled objects relies on GPT-4 (1-shot) to extract 3 relations: initial possessions, swaps, and the target person whose final possessed object is expected as the answer. Our reasoning program iterates through all the swaps starting from the initial state and retrieves the last possessed object associated with the target.

**Kinship reasoning (KR).** CLUTRR (Sinha et al. 2019) is a kinship reasoning dataset of stories which indicate the kinship between characters, and requires the model to infer the relationship between two specified characters. The questions have different difficulty levels based on the length of the reasoning chain, denoted by \( k \in \{2 \ldots 10\} \).

Our solution for kinship reasoning invokes GPT-4 (2-shot) to extract the kinship graph from the context. We also provide an external common-sense knowledge base for rules like “mother’s mother is grandmother”. Our program then uses the rules to derive other kinship relations. Lastly, we retrieve the kinship between the specified pair of people.

**Math reasoning (MR).** This task is drawn from the GSM8K dataset of arithmetic word problems (Cobbe et al. 2021). The questions involve grade school math word problems created by human problem writers, and the model is asked to produce a number as the result. Since the output can be fractional, we allow a small delta when comparing the derived result with the ground truth.

Our solution to this task prompts GPT-4 (2-shot) to produce step-by-step expressions, which can contain constants, variables, and simple arithmetic operations. We evaluate all the expressions through a DSL, and the result associated with the goal variable is returned. By focusing the LM’s responsibility solely on semantic parsing, our relational program can then achieve faithful numerical computation via DSL evaluation.

**Question answering with information retrieval (QA).** We choose HotpotQA (Yang et al. 2018), a Wikipedia-based question answering (QA) dataset under the “distractor” setting. Here, the model takes in 2 parts of inputs: 1) a question, and 2) 10 Wikipedia paragraphs as the context for answering the question. Among the 10 Wikipedia pages, at most 2 are relevant to the answer, while the others are distractions.

Our solution is an adaptation of FE2H (Li, Lei, and Yang 2022), which is a 2-stage procedure. First, we turn the 10 documents into a vector database by embedding each document. We then use the embedding of the question to retrieve the 2 most related documents, which are then fed to a language model to do QA. In this case, the QA model does not have to process all 10 documents, leading to less distraction.

**Product search (PS).** We use Amazon’s ESCI Product Search dataset (Reddy et al. 2022). The model is provided with a natural language (NL) query and a list of products (23
Our solution for CLEVR is similar, directly replicating the DSL provided by the original work. OWL-ViT and CLIP are used to detect objects and infer attributes, while the spatial relations are directly computed using the bounding box data.

**Visual object tagging (VOT).** We evaluate on two datasets, VQAR (Huang et al. 2021) and OFCP. For VQAR, the model is given an image and a programmatic query, and is asked to produce bounding boxes of the queried objects in the image. Our solution composes a relational knowledge base, defining entity names and relationships, with object retrieval (OWL-ViT) and visual QA (ViLT) models.

Online Faces of Celebrities and Politicians (OFCP) is a self-curated dataset of images from Wikimedia Commons among other sources. For this dataset, the model is given an image with a descriptive NL filename, and needs to detect faces relevant to the description and tag them with their names. Our solution obtains a set of possible names from GPT-4 and candidate faces from DSFD. These are provided to CLIP for object classification, after which probabilistic reasoning filters the most relevant face-name pairs.

**Language-guided image generation and editing (IGE).** We adopt the task of image editing from (Gupta and Kembhavi 2022). In this task, the instruction for image editing is provided through NL, and can invoke operations such as blurring background, popping color, and overlaying emojis. Due to the absence of an existing dataset, we repurpose the OFCP dataset by introducing 50 NL image editing prompts. Our solution for this task is centered around a DSL for image editing. We incorporate GPT-4 for semantic parsing, DSFD for face detection, and CLIP for entity classification. Modules for image editing operations are implemented as individual foreign functions.

For free-form generation and editing of images, we curate IGP20, a set of 20 prompts for image generation and editing. Instead of using the full prompt, we employ an LM to decompose complex NL instructions into simpler steps. We define a DSL with high-level operators such as generate, reweight, refine, replace, and negate. We use a combination of GPT-4, Prompt-to-Prompt (Hertz et al. 2022), and diffusion model (Rombach et al. 2022) to implement the semantics of our DSL. We highlight our capability of grounding positive terms from negative phrases, which enables handling prompts like “replace apple with other fruits” (Fig. 3).

**Experiments and Analysis**

We aim to answer the following research questions:

**RQ1.** Is VIEIRA programmable enough to be applicable to a diverse range of applications with minimal effort?

**RQ2.** How do solutions using VIEIRA compare to other baseline methods in the no-training setting?

**RQ1: Programmability**

While a user study for VIEIRA’s programmability is out of scope in this paper, we qualitatively evaluate its programmability on three aspects. First, we summarize the lines-of-code (LoC) for each of our solutions in Table 2. The programs

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th>#Test Samples</th>
<th>Metric</th>
<th>Foundation Models Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>DR</td>
<td>369</td>
<td>EM</td>
<td>GPT-4</td>
</tr>
<tr>
<td>TSO</td>
<td>TSO</td>
<td>150</td>
<td>EM</td>
<td>GPT-4</td>
</tr>
<tr>
<td>KR</td>
<td>CLUTRR</td>
<td>1146</td>
<td>EM</td>
<td>GPT-4</td>
</tr>
<tr>
<td>MR</td>
<td>GSM8K</td>
<td>1319</td>
<td>EM</td>
<td>GPT-4</td>
</tr>
<tr>
<td>QA</td>
<td>Hotpot QA</td>
<td>1000</td>
<td>EM</td>
<td>GPT-4</td>
</tr>
<tr>
<td>PS</td>
<td>Amazon ESCI</td>
<td>1000</td>
<td>nDCG</td>
<td>GPT-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ada-002</td>
</tr>
<tr>
<td>VQA</td>
<td>CLEVR</td>
<td>480</td>
<td>Recall@1</td>
<td>ViLT</td>
</tr>
<tr>
<td></td>
<td>GQA</td>
<td>500</td>
<td>Recall@3</td>
<td>OWL-ViT</td>
</tr>
<tr>
<td>VOT</td>
<td>VQAR</td>
<td>100</td>
<td>MI</td>
<td>OWL-ViT</td>
</tr>
<tr>
<td></td>
<td>OFCP</td>
<td>50</td>
<td>MI</td>
<td>DSFD</td>
</tr>
<tr>
<td>IGE</td>
<td>OFCP</td>
<td>50</td>
<td>MI</td>
<td>CLIP</td>
</tr>
<tr>
<td></td>
<td>IGP20</td>
<td>20</td>
<td></td>
<td>GPT-4</td>
</tr>
</tbody>
</table>

Table 1: Characteristics of benchmark tasks including the dataset used, its size, and evaluation metrics. Metrics include exact match (EM), normalized discounted cumulative gain (nDCG), and manual inspection (MI). We also denote the foundation models used in our solution for each task.

products on average). The goal is to rank the products that best match the query. In the dataset, for each pair of query and product, a label among $E$ (exact match), $S$ (substitute), $C$ (complementary), and $I$ (irrelevant) is provided. The metric we use to evaluate the performance is nDCG. The gains are set to be 1.0 for $E$, 0.1 for $S$, 0.01 for $C$, and 0.0 for $I$.

One challenge of this dataset is that many queries contain negative statements. For example, in the query “#1 treadmill without remote”, the “remote” is undesirable. Therefore, instead of computing the embedding of the full query, we decompose the query into positive and negative parts. We then perform semantic search by maximizing the similarity of the positive part while minimizing that of the negative part.

**Compositional visual question answering (VQA).** We choose two compositional VQA datasets, GQA (Hudson and Manning 2019) and CLEVR (Johnson et al. 2016). In this task, the model is given an image and a question, and needs to answer the question. For GQA, the majority of questions expect yes/no answers, while CLEVR’s questions demand features like counting and spatial reasoning. We uniformly sample 500 and 480 examples from GQA and CLEVR datasets respectively. Following VQA conventions (Kim, Son, and Kim 2021), we use Recall@k where $k \in \{1, 3\}$ as the evaluation metrics.

Our solution for GQA is an adaptation of VisPROG (Gupta and Kembhavi 2022). We create a DSL for invoking vision modules such as ViLT and OWL-ViT, and use GPT-4 for converting questions into programs in this DSL.
Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

What is the date one year ago from today in MM/DD/YYYY?

Today's meeting is rescheduled to 11 am tomorrow, 10/16/1924. Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.

Let's think step by step. First, we need to subtract one year from the given date, 10/16/1924. When we do this, we get 10/16/1923.
Table 5: Quantitative results on the VQA datasets.

<table>
<thead>
<tr>
<th>Method</th>
<th>GQA</th>
<th>CLEVR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall@1</td>
<td>Recall@3</td>
</tr>
<tr>
<td>ViLT-VQA</td>
<td>0.049</td>
<td>0.462</td>
</tr>
<tr>
<td>PNP-VQA</td>
<td>0.419</td>
<td>—</td>
</tr>
<tr>
<td>Ours</td>
<td>0.579</td>
<td>0.665</td>
</tr>
</tbody>
</table>

Table 6: Quantitative results on object tagging and image editing tasks. We manually evaluate the tagged entities and the edited images for semantic correctness rates.

<table>
<thead>
<tr>
<th>Method</th>
<th>Visual Object Tagging</th>
<th>Image Editing</th>
</tr>
</thead>
<tbody>
<tr>
<td>VQAR</td>
<td>67.61%</td>
<td>OFCP</td>
</tr>
<tr>
<td>OFCP</td>
<td>60.82%</td>
<td>OFCP</td>
</tr>
<tr>
<td>Ours</td>
<td>67.61%</td>
<td>60.82%</td>
</tr>
</tbody>
</table>

**Conclusion**

We introduced **VIEIRA**, a declarative framework designed for relational programming with foundation models. **VIEIRA** brings together foundation models from diverse domains, providing a unified interface for composition and the ability to perform probabilistic logical reasoning. This results in solutions with comparable and often superior performance than neural-based baselines. In the future, we aim to extend the capabilities of **VIEIRA** beyond the current in-context learning settings to weakly-supervised training and fine-tuning of foundation models in an end-to-end manner.

**Acknowledgements**

We thank the anonymous reviewers for useful feedback. This research was supported by NSF grant #2313010 and DARPA grant #FA8750-23-C-0080. Ziyang Li was supported by an Amazon Fellowship.
References


