A Declarative System for Optimizing AI Workloads

A Preprint

Chunwei Liu*, Matthew Russo*, Michael Cafarella,
Lei Cao[†], Peter Baille Chen, Zui Chen, Michael Franklin[‡],
Tim Kraska, Samuel Madden, Gerardo Vitagliano

MIT, [†]University of Arizona, [‡]University of Chicago

chunwei@mit.edu, mdrusso@mit.edu, michjc@csail.mit.edu
caolei@arizona.edu, peterbc@mit.edu, chenz429@mit.edu, mjfranklin@uchicago.edu,
kraska@mit.edu, madden@csail.mit.edu, gerarvit@csail.mit.edu

Semantic Analytics Applications

A homebuyer wants to use online real estate listing data to find a place that is

(a)modern and attractive

(b)within two miles of MIT

A researcher in cancer research domain would like to

(a) download a dozen of cancer research papers

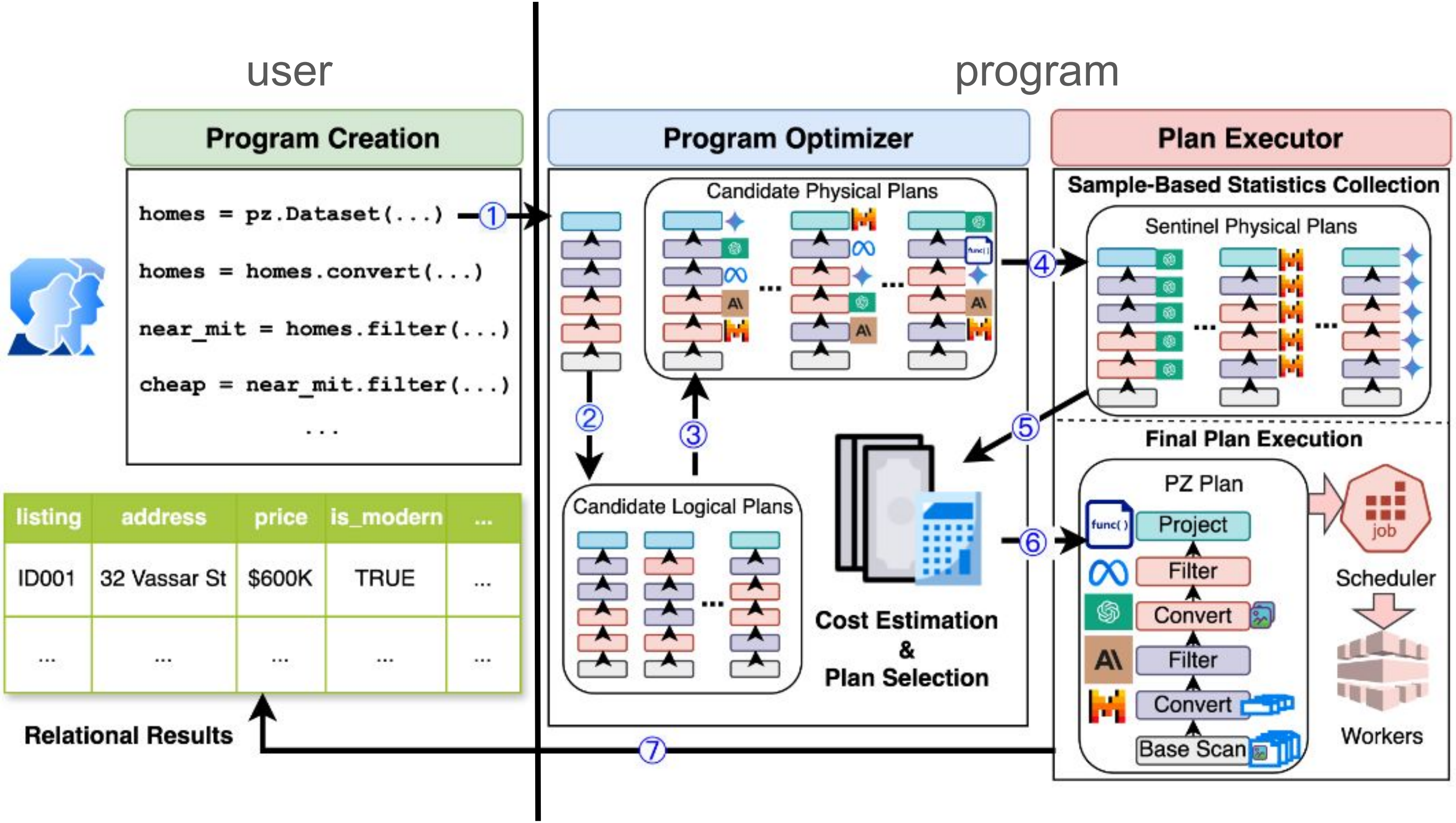
(b)identify the papers that contain patient experiment data

(c)integrate those data into a single table

Semantic Analytics Applications

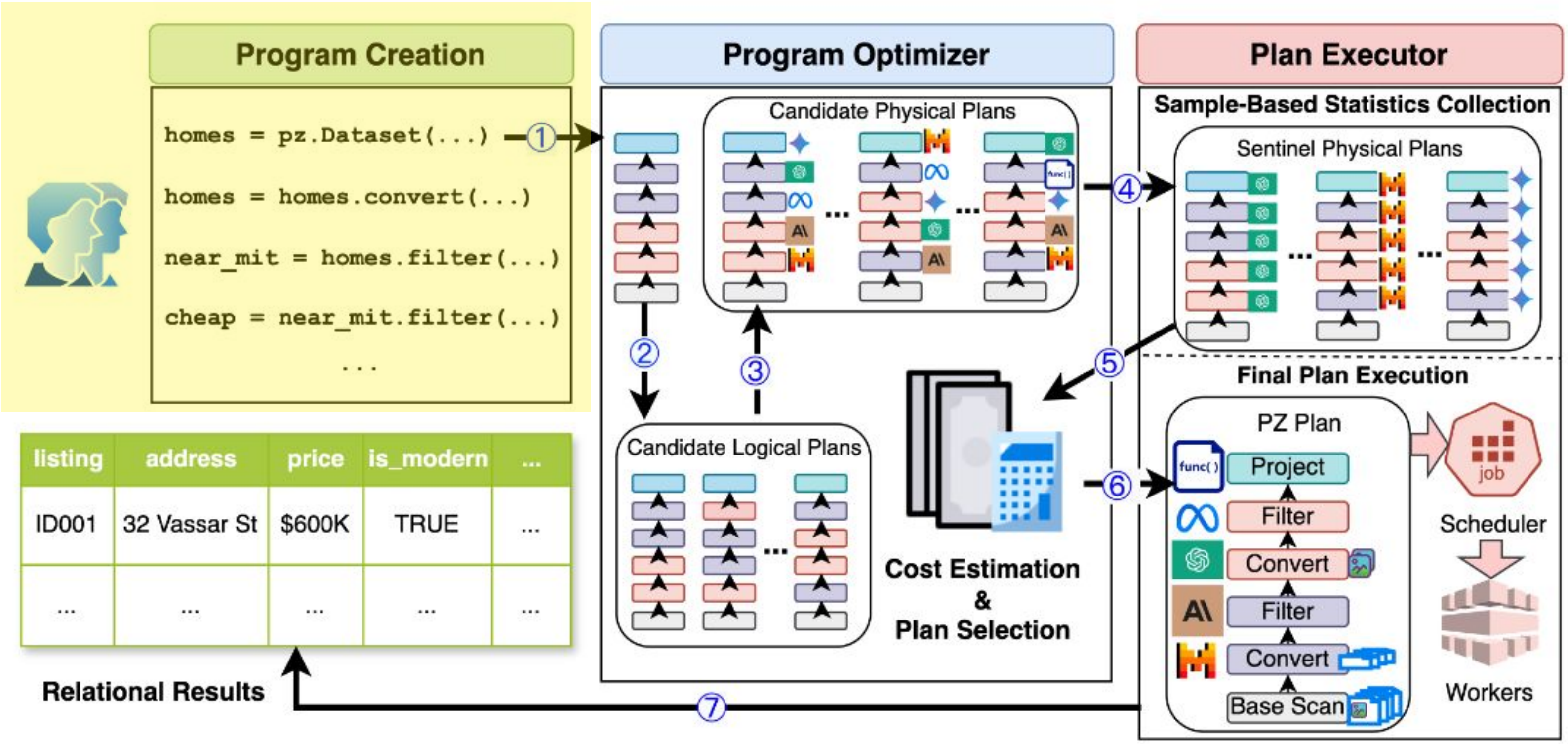
- Traditional data processing + AI-like semantic reasoning
- Data-intensive
- Can be decomposed into an execution tree of distinct operations
- May result in answers of varying quality

Overview



user
program
Program Creation
homes = pz.Dataset(...)
homes = homes.convert(...)
near_mit = homes.filter(...)
cheap = near_mit.filter(...)
...
Program Optimizer
Candidate Physical Plans
Candidate Logical Plans
Cost Estimation & Plan Selection
Plan Executor
Sample-Based Statistics Collection
Sentinel Physical Plans
Final Plan Execution
PZ Plan
func( )
Project
Filter
Convert
Filter
Convert
Base Scan
job
Scheduler
Workers
listing | address | price | is_modern | ...
ID001 | 32 Vassar St | $600K | TRUE | ...
Relational Results
1
2
3
4
5
6
7

Program Creation



Custom schema in a relational view

create a custom schema for the input dataset of Emails

```python
import palimpzest as pz

class Email(pz.TextFile):
  """Represents an email, which can subclass a text file"""
  sender = pz.StringField(desc="The email address of the sender", required=True)
  subject = pz.StringField(desc="The subject of the email", required=True)

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

# user specified policy
policy = pz.MinimizeCostAtFixedQuality(min_quality=0.8)

# execute plan
results = pz.Execute(emails, policy=policy)
```

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

```python
import palimpzest as pz

class Email(pz.TextFile):
  """Represents an email, which can subclass a text file"""
  sender = pz.StringField(desc="The email address of the sender", required=True)
  subject = pz.StringField(desc="The subject of the email", required=True)

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

# user specified policy
policy = pz.MinimizeCostAtFixedQuality(min_quality=0.8)

# execute plan
results = pz.Execute(emails, policy=policy)
```

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

Natural language descriptions

- For developers to specify correct program output
- For systems to find a high-quality implementation
- No need for prompt engineering from user side
- What if the given description contains error?
    - Skip this record processing
- What if it’s ambiguous?
    - Explained in correctness section

Logical relational operators

Convert

- Transform a typed data object into a new object with specified schema
- The user does not need to specify how to implement a convert operation
- If LLM is needed,

operator	description
Project	π(rel., cols)
Select	σ(rel., predicate)
Convert	χ(rel., schema_a, schema_b)
Group By	Γ(rel., group_cond., agg.)
Limit	L(rel., limit)
Agg.	α(rel., agg_func)

(b) PALIMPZEST's full relational algebra. We extend the traditional relational algebra to include operators such as groupby which produce multiple relations.

```python
import palimpzest as pz

class Email(pz.TextFile):
  """Represents an email, which can subclass a text file"""
  sender = pz.StringField(desc="The email address of the sender", required=True)
  subject = pz.StringField(desc="The subject of the email", required=True)

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

# user specified policy
policy = pz.MinimizeCostAtFixedQuality(min_quality=0.8)

# execute plan
results = pz.Execute(emails, policy=policy)
```

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

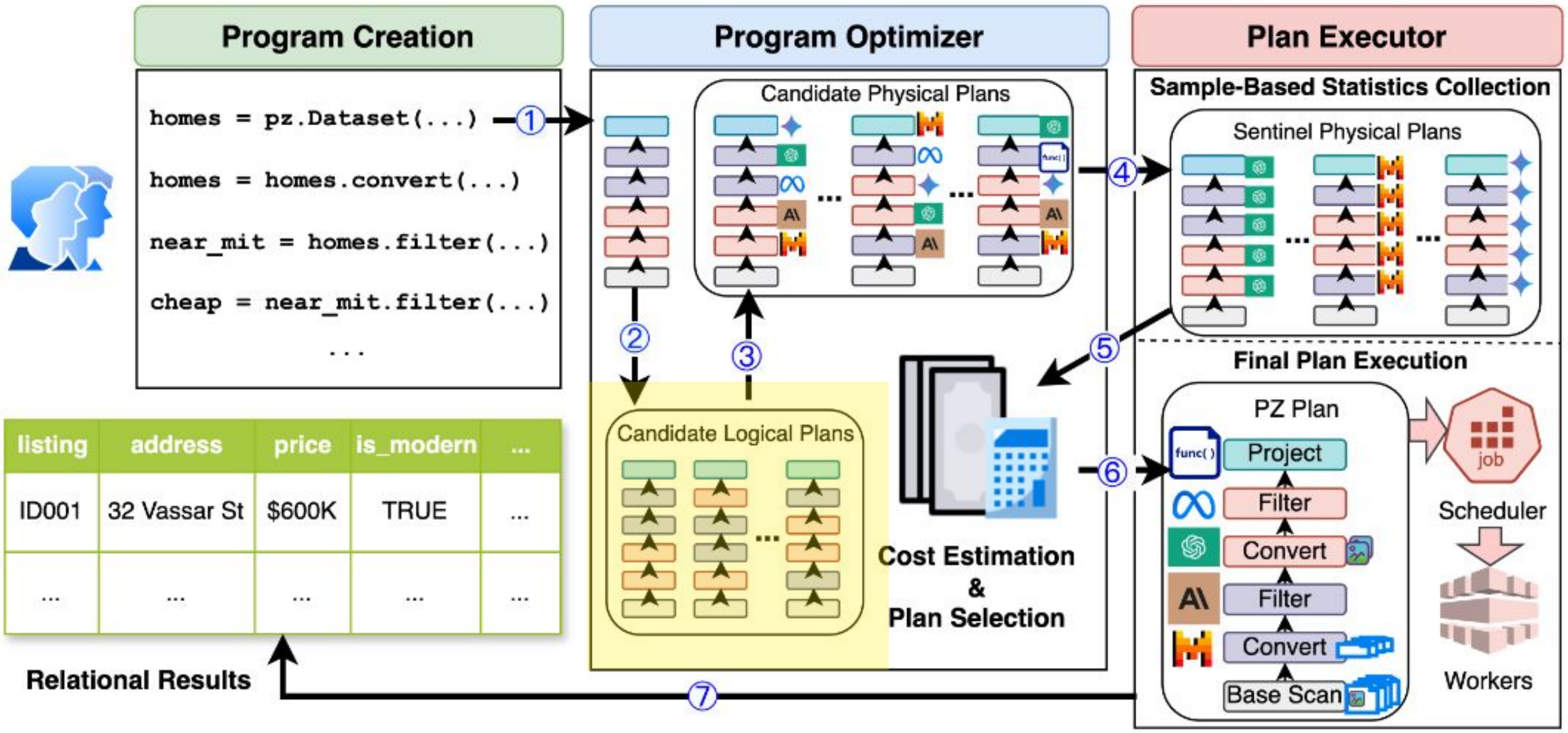
Correctness & Quality

- Correctness
  - What if the description is ambiguous?
  - Modify to be more precise
  - Split into more concrete steps
  - Future: validation example from user side
- Quality
  - How to assess the quality of an execution plan?
  - How to find approaches to maximize assessed data quality?
  - Compare with GPT-4 result as evaluation in experiment

```python
import palimpzest as pz

class Email(pz.TextFile):
  """Represents an email, which can subclass a text file"""
  sender = pz.StringField(desc="The email address of the sender", required=True)
  subject = pz.StringField(desc="The subject of the email", required=True)

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

# user specified policy
policy = pz.MinimizeCostAtFixedQuality(min_quality=0.8)

# execute plan
results = pz.Execute(emails, policy=policy)
```

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

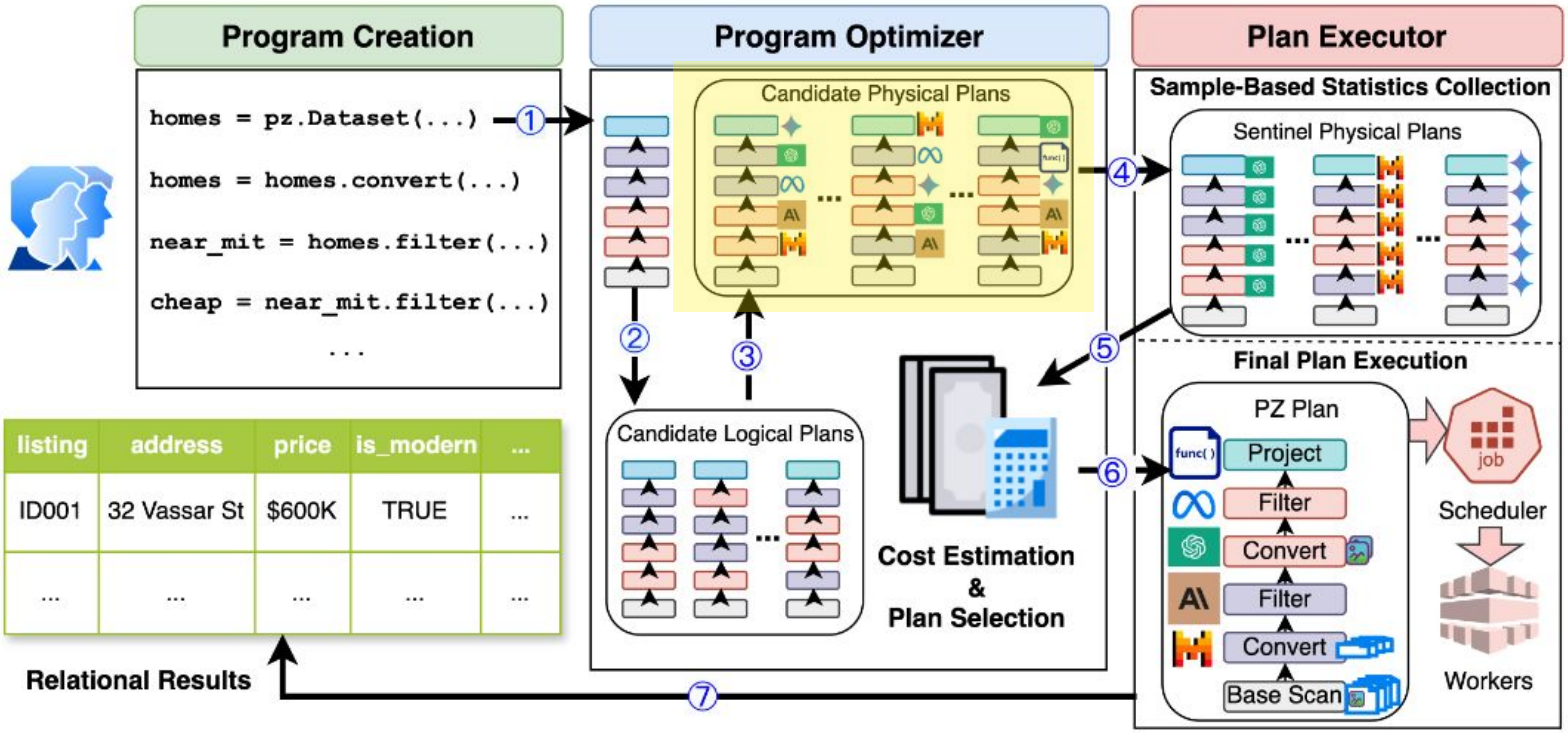
Program Optimizer



Logical Plan Optimization

- Filter reordering
  - Permutes the ordering of selection filters (3 steps -> 6 plans)
- Convert reordering
  - Move “convert” operations around the logical plan

Program Optimizer



Physical Plan Optimization Methods

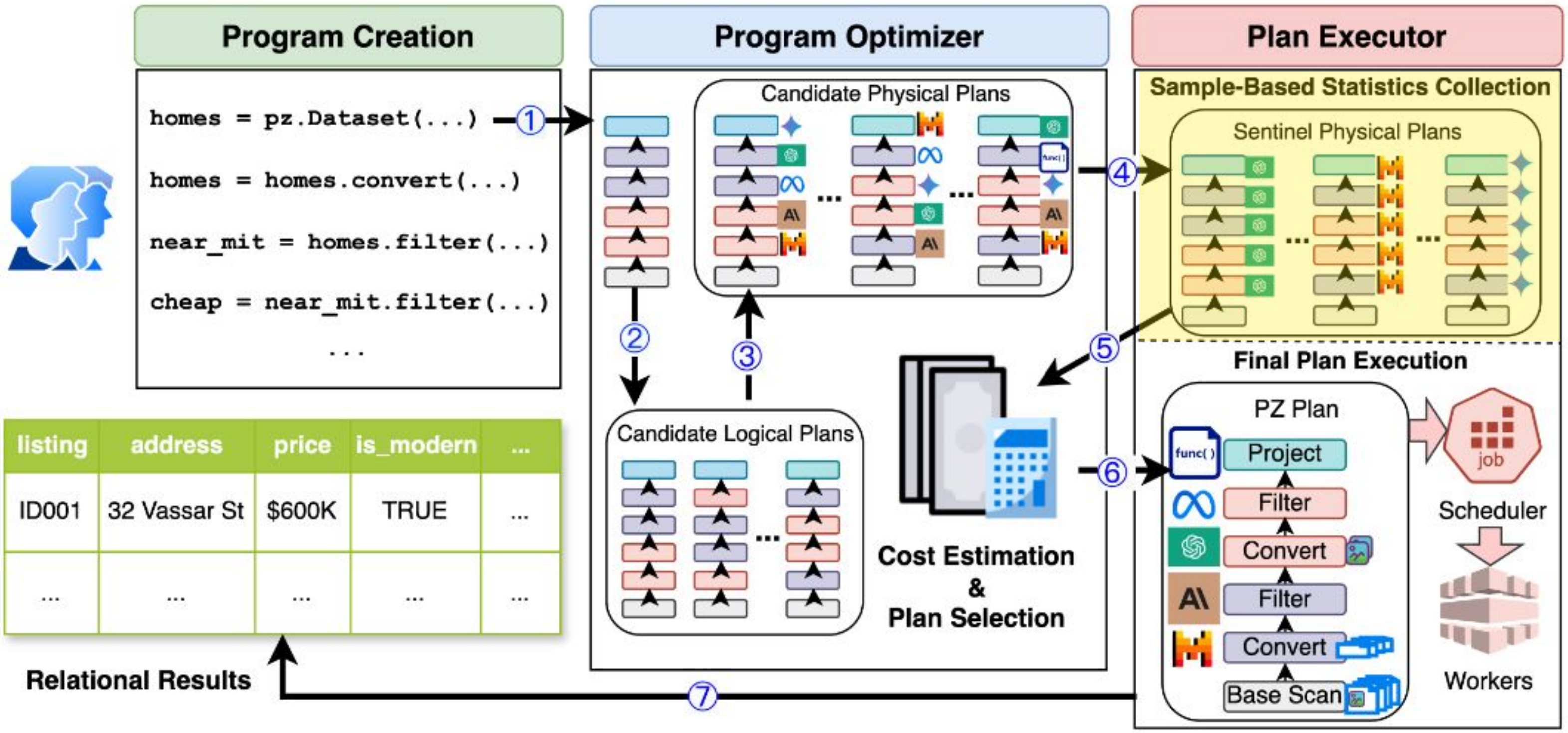
Implemented

- Model selection
- Code synthesis
- Multi-data prompt marshalling
- Input token reduction

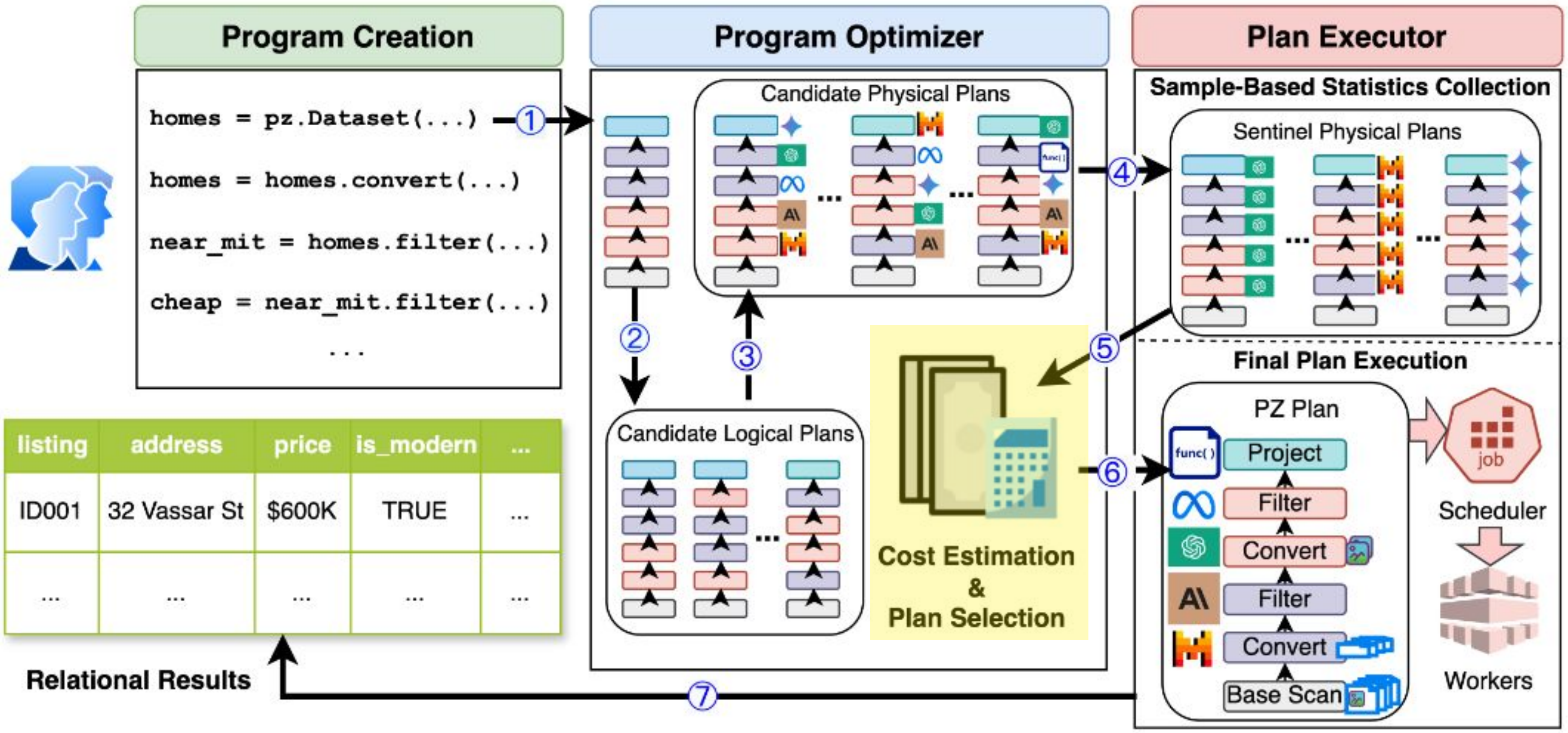
Future plan

- Output token reduction
- Model cascades
- Knowledge distillation
- Workload-aware execution management

Sample based Statistics Collection



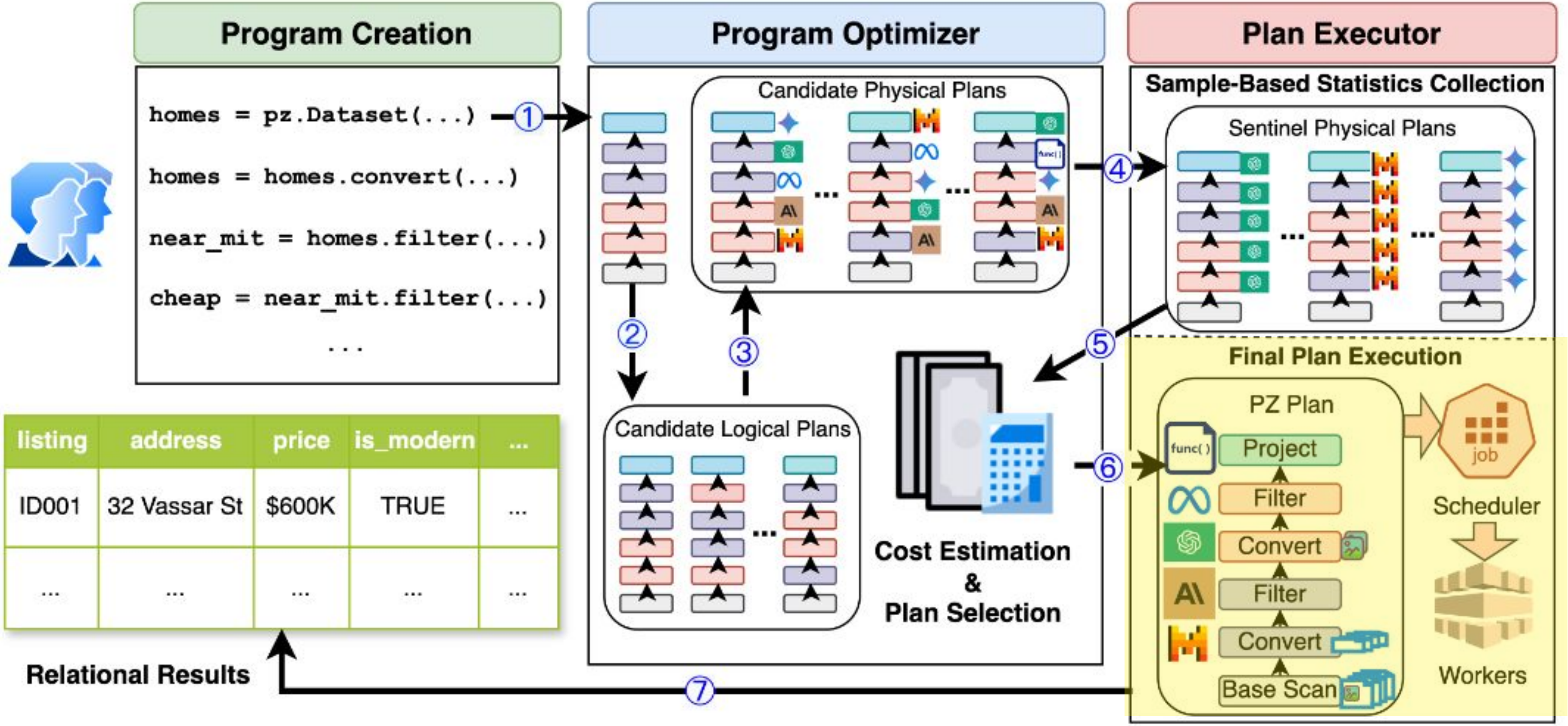
Cost Estimation & Plan Selection



Cost Estimation & Plan Selection

Algorithm 1 Optimized Plan Selection Algorithm

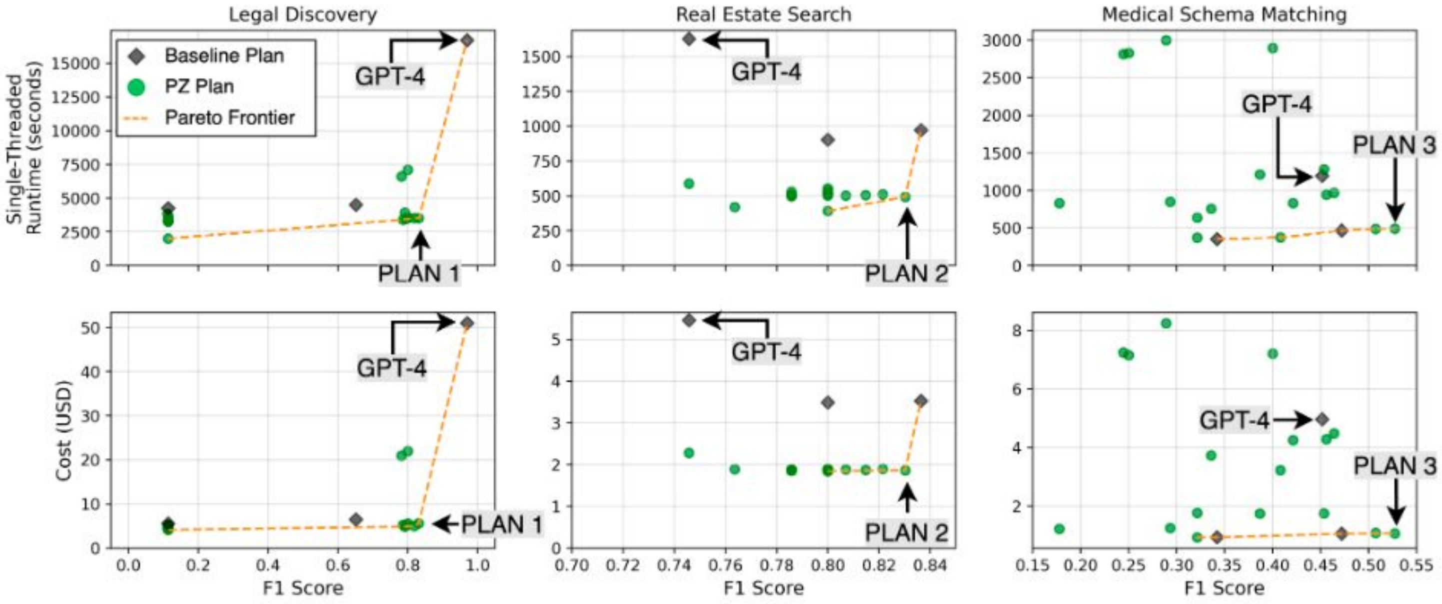
```
Require: userCode, userPolicy                                                      # Step 1
logicalPlans = generateLogicalCandidates(userCode)                                 # Step 2
sentinelPhysicalPlans = getPhysicalPlans(logicalPlans, sentinel = True)            # Step 3

performanceStatisics = {}
for 0...NUM_SAMPLES do
    input = getSampledInput()
    stats = runAndComputeStatistics(sentinelPhysicalPlans, input)                  # Step 4
    performanceStatistics.update(stats)                                            # Step 5
end for

physicalCandidates = getPhysicalPlans(logicalPlans, stats = performanceStatistics)
reducedCandidates = naiveElimination(physicalCandidates)
frontierCandidates = scoreAndEliminatePlans(reducedCandidates, performanceStatistics)

return chooseBestPlan(frontierCandidates, userPolicy)                              # Step 6
```



Program Creation
homes = pz.Dataset(...)
homes = homes.convert(...)
near_mit = homes.filter(...)
cheap = near_mit.filter(...)
...
Program Optimizer
Candidate Physical Plans
Candidate Logical Plans
Cost Estimation & Plan Selection
Plan Executor
Sample-Based Statistics Collection
Sentinel Physical Plans
Final Plan Execution
PZ Plan
Project
Filter
Convert
Filter
Convert
Base Scan
job
Scheduler
Workers
listing | address | price | is_modern | ...
ID001 | 32 Vassar St | $600K | TRUE | ...
Relational Results

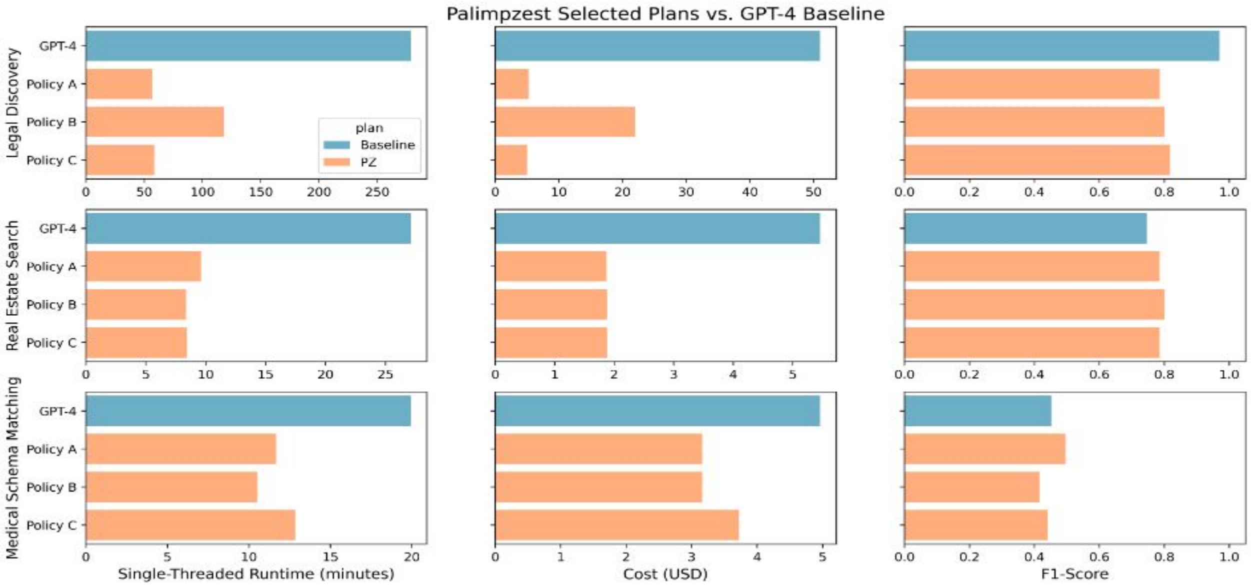
Experiments - 1

- Palimpzest can generate plans which provide users with compelling performance trade-offs



Experiments - 2

- Palimpzest can identify plans that have better end-to-end runtime, cost and quality than a naive plan that uses the same state-of-the-art language model for each operation



Experiments - 3

- They ran Palimpzest with parallel implementations of the convert and filter operations to demonstrate the system's ability to achieve large runtime speedups

(a) Legal Discovery

Plan	Runtime (s)	Cost ($)	F1
Single-Threaded Baseline (GPT-4)	16,712	51.0	0.97
Palimpzest	185 (1.1%)	5.60 (11.0%)	0.81 (83.5%)

(b) Real Estate Search

Plan	Runtime (s)	Cost ($)	F1
Single-Threaded Baseline (GPT-4)	1,626	5.46	0.75
Palimpzest	80.9 (5.0%)	1.86 (34.1%)	0.80 (107%)

(c) Medical Schema Matching

Plan	Runtime (s)	Cost ($)	F1
Single-Threaded Baseline (GPT-4)	1,195	4.96	0.45
Palimpzest	215 (18.0%)	3.36 (67.7%)	0.46 (102%)