GIO: Generating Efficient Matrix and Frame Readers for Custom Data Formats by Example

Saeed Fathollahzadeh[1,2]
Matthias Boehm[3]

1 TU Graz 2 Know Center 3 TU berlin

June 18–23
Seattle, WA, USA

SIGMOD PODS 2023

Motivation

- Data formats vary in structure, syntax, semantics, and compression
  Examples: CSV, JSON, XML, Parquet, ORC, HDF5, FITS, etc.
- Existing systems have limited support for custom formats
- Custom readers require low-level programming and system knowledge
- Custom readers are not portable across systems and languages

➡ Having state-of-the-art readers isn't enough:
Is there an efficient and automatic way to get around?

Problem of Custom Data Formats

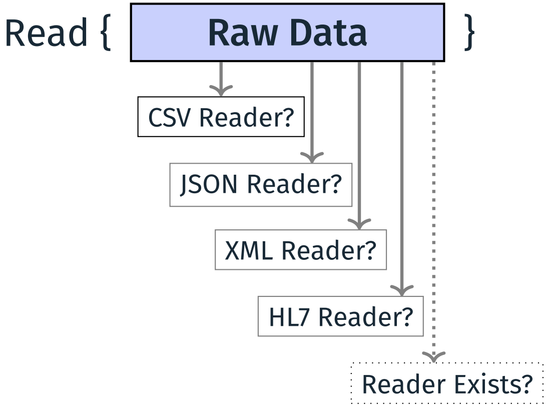
Custom data formats originate from systems and machines, whose data representation was not designed for data **exchange** and **interoperability**

- Such data formats include flat or nested structure
- Optional key or positional attributes
- Multiple custom delimiters or prefixes
- Potentially multi-line records
- Undocumented semantics of attribute sequences
- Co-appearances and repeating groups of attributes

Examples: semiconductor manufacturing, smart grid data management, paper production, and waste recycling

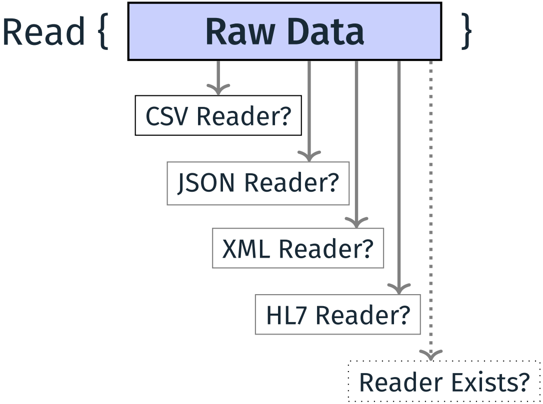
Is there an automatic way to get around?

Today:

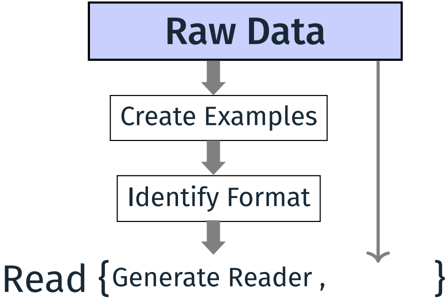


Is there an automatic way to get around?

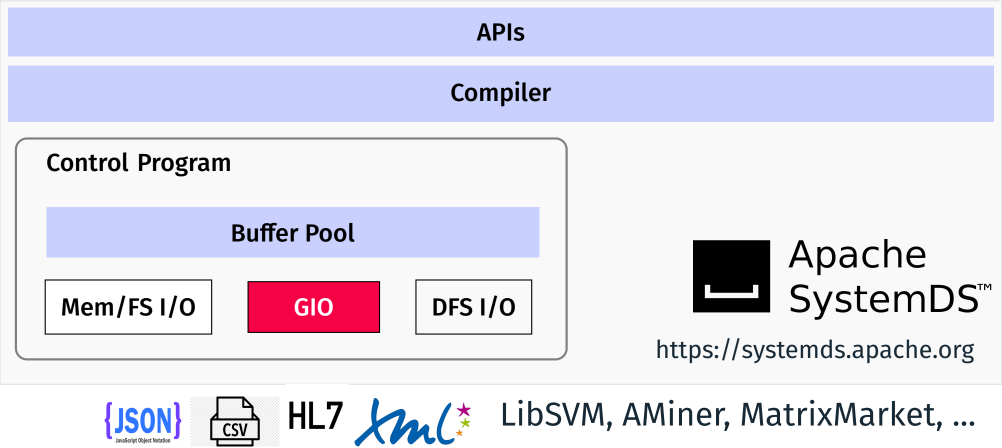
Today:



GIO: Generating Efficient Readers



SystemDS I/O Gen API



GIO's Parameters

Given a custom text-based dataset **D**, and user-provided examples of **raw** and **target data**:

- **Sample Raw (S) Input:** Let $S = \{S_1, S_2, \ldots, S_l\}$ be a list of input strings (i.e., selected rows of the input dataset D).
- **Sample Matrix/Frame (F) Input:** Let $F = \{F_1, F_2, \ldots, F_n\}$ be a sample matrix or frame, corresponding to S with n records.

➡ our goal is to generate a matrix or frame reader for reading the full dataset and other data of this format.

Example Parameters

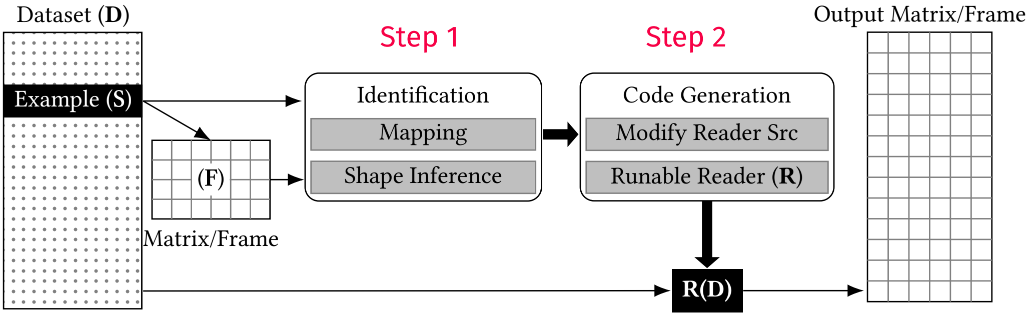
```
#index 2015101   ← beginning of record #1
#* NoDB: efficient query execution on raw data files
#@ Ioannis Alagiannis; Renata Borovica; Anastasia Ailamaki
#o EPFL Switzerland; EPFL Switzerland; EPFL Switzerland
#t 2015
#c VLDB Conference
#% ... unlimited set of references
#! As data collections become larger and larger, data loading evolves to a major bottleneck. ...
#index 2019102   ← beginning of record #2
#* Pangea: Monolithic Distributed Storage for Data Analytics
#@ Jia Zou; Arun Iyengar; Chris Jermaine
#o Rice University; Rice University; Rice University
#t 2019
#c VLDB Conference
#% ... unlimited set of references
#! Storage and memory systems for modern data analytics are heavily layered, managing shared ...
#index 2018103   ← beginning of record #3
#* Filter Before You Parse: Faster Analytics on Raw Data
#@ Shoumik Palkar; Firas Abuzaid; Matei Zaharia
#o Stanford InfoLab; Stanford InfoLab; Databricks Inc
#t 2018
#c VLDB Endowment
#% ... unlimited set of references
#! Exploratory big data applications often run on raw unstructured or semi-structured data ...
```

Sample Raw (S)

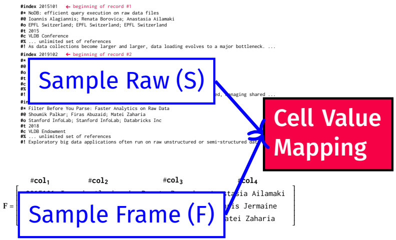
$$\begin{array}{cc} & \begin{array}{cccc} \#\mathbf{col}_1 & \#\mathbf{col}_2 & \#\mathbf{col}_3 & \#\mathbf{col}_4 \end{array} \\ \mathbf{F} = & \begin{bmatrix} \texttt{2015101} & \texttt{Ioannis Alagiannis} & \texttt{Renata Borovica} & \texttt{Anastasia Ailamaki} \\ \texttt{2019102} & \texttt{Jia Zou} & \texttt{Arun Iyengar} & \texttt{Chris Jermaine} \\ \texttt{2018103} & \texttt{Shoumik Palkar} & \texttt{Firas Abuzaid} & \texttt{Matei Zaharia} \end{bmatrix} \end{array}$$

Sample Frame (F)

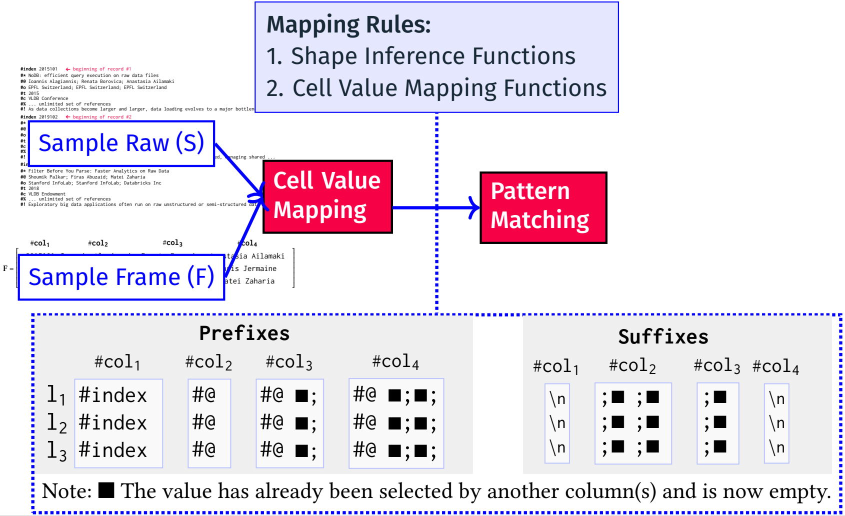
GIO Overview



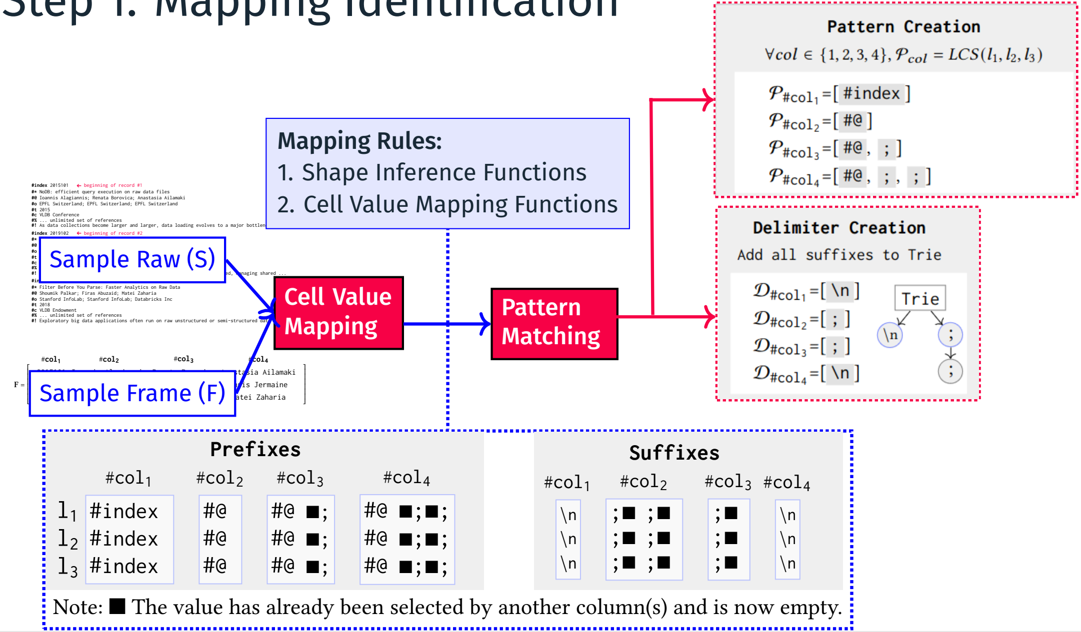
Step 1: Mapping Identification



Step 1: Mapping Identification



Step 1: Mapping Identification



Step 2: Code Generation

- Composed according to the passed mapping rules.
- Template-based Code Generation: **parsing primitives, conditions, path expressions**, and **value indexing**:
  - Code Templates:
    - pre-pass for obtaining additional metadata from data
    - inferring the dimensions
    - iterate over records of the raw dataset
  - Indexing Code:
    - **Row Indexing Code:** determine the number of rows
    - **Column Index and Value Code:** code gets column info and values
  - Cell Value Code:
    - **Cell Value by Nested Conditions**
    - **Cell Value by Sequential String Matching**
    - **Cell Value by Regular Expressions**

Overview of GIO Code Templates and Code Generation

GenericTemplate($\mathcal{R}$, $\mathcal{C}$, $\mathcal{V}$)

```
[srcInitRow, srcRow] = GenerateRowC
[srcInitCol, srcCol] = GenerateColC
src = " InitFile(); " +
      " FrameBlock ReadBlock( Buffer
      " pro = Estimation(br, 𝓡, 𝓒,
      " FrameBlock fb = new FrameBlo
        srcInitRow + // row pre-
        srcInitCol + // col pre-
      " for each (record r in br) {
        srcRow + // row index co
        srcCol + // extract and
      " } "
      " return fb; } ";
return src;
```

Details are in the paper!

GenerateRowCode($\mathcal{R}$)

```
srcInit = "  ";    src = "  ";
if (𝓡_structure == Identity)
  srcInit = " if (𝓡_pattern ∈ r) "+
            "   rowIndex++; ";
else if (𝓡_structure == Constraint || 𝓡_structure == Max)
  srcInit = " _index = 0; " +
            " for (key k ∈ 𝓡_pattern) " +
            "   _index = r.IndexOf(k, _index); " +
            " _end = r.IndexOf(𝓡_delimiter, _index); " +
            " _text = r.Substring(_index, _end); " +
            " rowIndex = ParseInt(_text); ";
else if (𝓡_structure == Stack)
  srcInit = " _bList = [];   _eList = []; " +
            " for (record r ∈ br) { " +
            "  _index = 0;   _end = 0; " +
            "  while (_index != 0) { " +
            "   _index = r.IndexOf(𝓡_pattern[0], _index);
            "   if(_index != 0)  " +
            "    _blis.append(Pair(r.index, _index));
            "  while (_end != 0) { " +
            "   _end = r.IndexOf(𝓡_pattern[1], _end); " +
            "   if(_end != 0)  " +
            "    _eList.append(Pair(r.index, _end));
            " _rIndexes = [];    _stack = Stack(); " +
            " for (int i=0, j=0; i < min(len(_bList),
            "  if(_bListi] < _eList[j])   _stack.push(
```

GenerateColCode($\mathcal{V}$, $\mathcal{T}$)

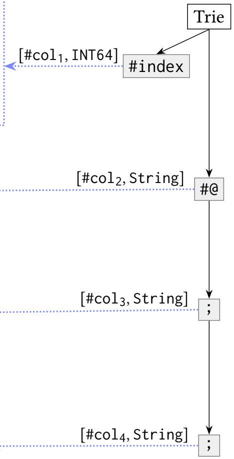
```
srcInit = "  ";    src = "  ";
trie = InitTrie( Root );
for ( column v ∈ 𝓥)
  Node node = new Node( 𝓥_key, 𝓥_colIndex, 𝓥_valueType );
  trie.insert( node ); // node's key is a list of strings
if 𝓣> |trie.Root.GetChild()|
 if trie.GetHeight() == |trie.GetNodes()|
   src = GenerateCodeRegular( trie );
 else
   src = GenerateCodeTrie( trie.Root, src, " 0 " );
else
 regexes = EmptySet();    map = EmptyMap();
 for ( column v ∈ 𝓥)
   regex = BuildRegex( v_key );    regexes.Add( regex );
   map.Put( v_key, v_colIndex );
 srcInit = MapToString( map ); // convert map to string src
 src = GenerateCodeRegex( regexes );
return srcInit, src;
```

GenerateCodeTrie(Node node, String pos)

```
if ( node ∈ EndOfColPattern )
 src += " _end = FindEndPos(" + node_delimiter+ "); " +
        " _text = r.Substring(" + pos + " , _end); " +
        " _val = Parse(_text, " + node.valType + " ); " +
        " fb.set(rowIndex, " + node.cellIndex + " , _val); "
if ( |node.GetChild()| > 0 )
 for ( child in node.GetChild() )
   src += " index = r.IndexOf(" + child_key +" , "+ pos+ " ); " +
          " if ( index != 0 ) { " +
          "  _newPos = _index+len( "+child_key+" ); "
   GenerateCodeTrie(child, newPos);
```

Reader Generation Example by Nested Conditions

```
_index = r.indexOf("#index", 0);
if(_index != -1) {
  _index += 6;
  _end = r.length();
  _text = r.substring(_index, _end);
  fb.set(rowIndex, 1, Parse(_text, INT64));
}
_index = r.indexOf("#@", 0);
if(_index != -1) {
  _index += 2;
  _end = r.indexOf(";", _index);
  _text = r.substring(_index, _end);
  fb.set(rowIndex, 2, _text);
    _index = r.indexOf(";", _end);
    if(_index != -1) {
      _index += 1;
      _end = r.indexOf(";", _index);
      _text = r.substring(_index, _end);
      fb.set(rowIndex, 3, _text);
    _index = r.indexOf(";", _index);
    if(_index != -1) {
      _index += 1;
      _end = r.length();
      _text = r.substring(_index, _end);
      fb.set(rowIndex, 3, _text);
    } } }
```



Pattern Creation

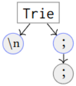
$\forall col \in \{1,2,3,4\}, \mathcal{P}_{col} = LCS(l_1, l_2, l_3)$

$\mathcal{P}_{\#col_1}$=[#index]
$\mathcal{P}_{\#col_2}$=[#@]
$\mathcal{P}_{\#col_3}$=[#@ , ;]
$\mathcal{P}_{\#col_4}$=[#@ , ; , ;]

Delimiter Creation

Add all suffixes to Trie

$\mathcal{D}_{\#col_1}$=[\n]
$\mathcal{D}_{\#col_2}$=[;]
$\mathcal{D}_{\#col_3}$=[;]
$\mathcal{D}_{\#col_4}$=[\n]



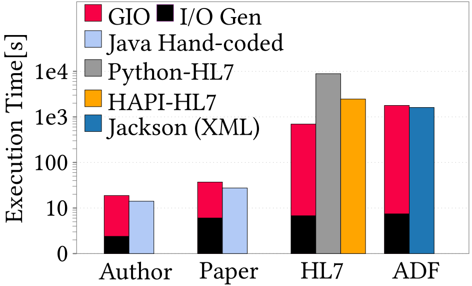
Experiments – Experimental Setting

- **HW:** AMD EPYC 7302 CPU @ 3.0-3.3 GHz (32 cores) 128 GB DDR4 RAM
- **SW:** Ubuntu 20.04.1, OpenJDK 11, Python 3.8, and Clang++10
- **Datasets:**

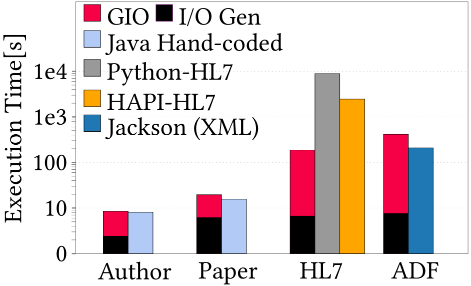
Dataset	n (nrow)	m (ncol)	o (objects)	Size [GB]
AMiner-Author (JSON)	1,712,432	Nested	1	0.62
AMiner-Paper (JSON)	2,092,355	Nested	2	3.7
Yelp (JSON)	8,635,403	Nested	7	19
AMiner-Author (Custom)	1,712,432	N/A	N/A	0.5
AMiner-Paper (Custom)	2,092,355	N/A	N/A	2.1
HL7 (Custom)	10,240,000	100	N/A	7.5
Yelp-Review (CSV)	8,635,403	9	Flat	6.5
Mnist8m (LibSVM)	8,100,000	784	Flat	12
Susy (LibSVM)	5,000,000	18	Flat	2.4
Higgs (CSV)	11,000,000	28	Flat	7.5
Queen (MM)	4,147,110	4,147,110	Flat	4.5
ReWaste F (CSV)	1,953,434	313	Flat	1.2
ADF (XML)	10,000,000	146	20	41

- **Baselines:** Apache SystemDS™ RapidJSON pandas { Gson } Jackson JSON HL7 <Hapi/> JSON4J

Reader Performance on Full Custom Datasets



(a) Single-threaded Readers

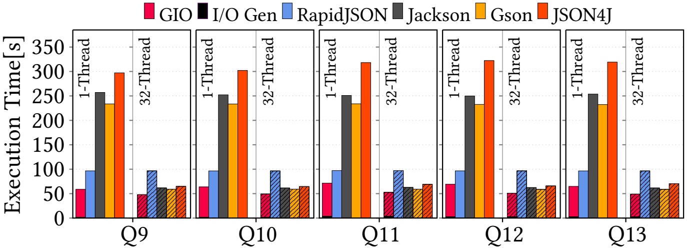


(b) Multi-threaded Readers

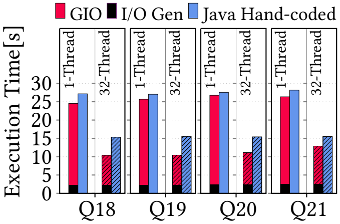
Micro-Benchmark Use Cases with Various Data/Query Characteristics

Q#	Dataset	Format	(Projection) Query	Nesting & Array
Q1	AMiner-Author	JSON	index	L1
Q2	AMiner-Author	JSON	name, paper_count	L1
Q3	AMiner-Author	JSON	index, name, paper_count, citation_number, hIndex	L1
Q4	AMiner-Author	JSON	name, affiliations[1, 2, 3, 4]	L1, L1 Array
Q5	AMiner-Paper	JSON	index	L1
Q6	AMiner-Paper	JSON	title, year	L1
Q7	AMiner-Paper	JSON	index, title, year, publication_venue, abstract	L1
Q8	AMiner-Paper	JSON	index, references[1, 2, 3, 4]	L1, L1 Array
Q9	Yelp	JSON	id	L1
Q10	Yelp	JSON	id, text	L1
Q11	Yelp	JSON	id, text, business.id, user.id, business.postal_code	L1, L2
Q12	Yelp	JSON	id, text, business.id, user.id, business.checkin.date, business.attribute.wifi	L1, L2, L3
Q13	Yelp	JSON	business.checkin.date, business.hours.monday, business.attribute.HhashTV	L3
Q14	AMiner-Author	Custom	index	N/A
Q15	AMiner-Author	Custom	name, paper_count	N/A
Q16	AMiner-Author	Custom	index, name, paper_count, citation_number, hIndex	N/A
Q17	AMiner-Author	Custom	name, affiliations[1, 2, 3, 4]	N/A
Q18	AMiner-Paper	Custom	index	N/A
Q19	AMiner-Paper	Custom	title, year	N/A
Q20	AMiner-Paper	Custom	index, title, year, publication_venue, abstract	N/A
Q21	AMiner-Paper	Custom	index, references[1, 2, 3, 4]	N/A
Q22	Yelp-Review	CSV	id	FLAT
Q23	Yelp-Review	CSV	id, text, stars	FLAT
Q24	HL7	Custom	evn_code, datetime, reason_code, operator_id	N/A
Q25	HL7	Custom	patient_name, birth_day, address, phone_number, account_number	N/A

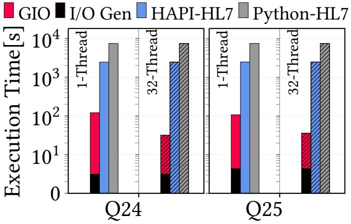
Micro Benchmark - Reader Runtime Performance



(a) Yelp (JSON)

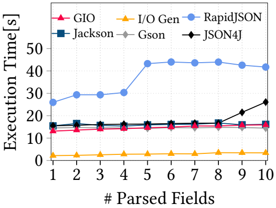


(b) AMiner-Paper (Custom)

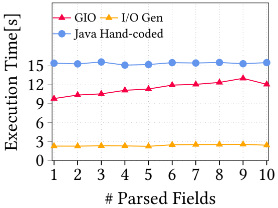


(c) HL7 (Custom)

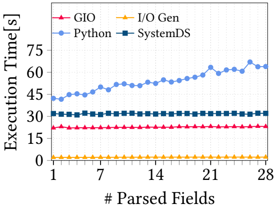
Comparison with Varying Number of Attributes



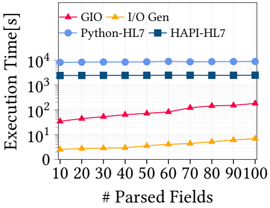
(a) Yelp (JSON)



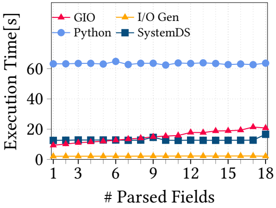
(b) AMiner Paper (Custom)



(c) Higgs (CSV)



(d) HL7 (Custom)



(e) Susy (LibSVM)

- ❑ RapidJSON is single-threaded and slow
- ❑ Projections are not exploited by SystemDS
- ❑ Python is doing projection, but it’s still slow
- ❑ Python reads sparse formats slowly
- ❑ GIO linear scales and robust performance

Conclusions

- Summary
  - GIO (generated I/O) reader framework for custom text data formats
  - GIO automatically identifies position/value mapping rules by giving samples
  - Efficiently generates code for efficient, multi-threaded readers for datasets
- Conclusion
  - GIO is capable of correctly identifying the mapping rules
  - Generated readers yield competitive performance
  - GIO makes data analysis and modeling easier with custom data formats
  - Users can adjust mapping rules and readers manually