Tensor Organizer (Torg): a metadata and query system for complex time series data

Matthew J. Bryan, Anuradha Ramachandran, Raman SV

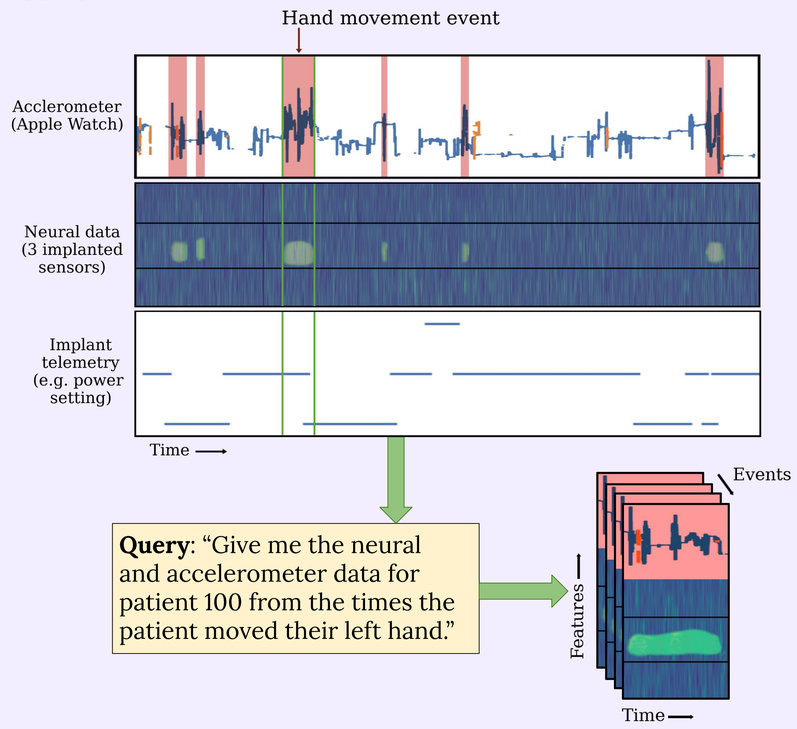
Summary

Torg indexes and queries multidimensional time series data across storage formats, delivering the results as tensors and timestamps. SQL stores the metadata and performs the queries.

Central concepts

	Example	Example metadata
Time series data Can be a tensor of arbitrary rank and numerical data type.	Neural data from a grid of 64 brain-implanted electrodes	{patientID: 100, sessionID: 1004, deviceManufacturer: “Medtronic”}
Events Events represent intervals on the timeline matching particular properties. Each refers to one or more time series.	The patient initiated a movement of their hand.	{whichHand: “left”}
Data class Every time series and event belongs to exactly one class. The metadata schema is the same for all instances of a class.	“neuralSensor”	{patientID: int, sessionID: int, deviceManufacturer: str}
Time series query Querying time series and events, optionally based on metadata. Returns a list of tensors and timestamps.	“Give me the neural data for patient 100, session 1004, for the 5 second interval around the time they moved their left hand”	time series filter: {class: “neuralSensor”, patientID: 100, sessionID: 1004} event filter: {class: “hand movement”, whichHand: “left”}

A real world example: neural engineering data



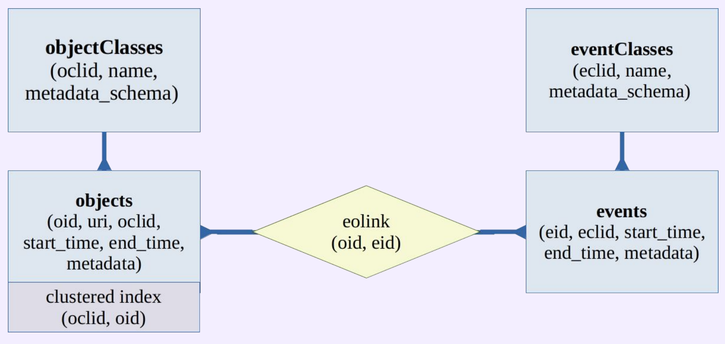
Code example: querying using an ORM

```
# 'loader' is an adapter for the hdf5 format
loader = H5demo.H5TensorLoader
sf = StaticFilter("Neural", metadata={"patientID": 100})
ef = EventFilter("hand movement", metadata={"whichHand": "left"})
# 'db' is our database adapter
events, event_timestamps = query.query(db, loader, static_filter=sf,
                                       event_filter=ef)
```

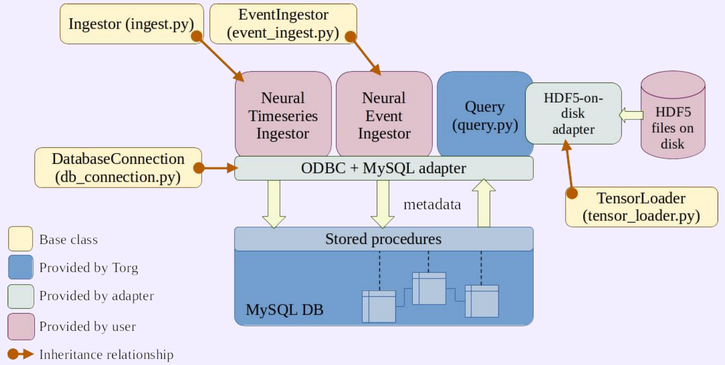
Generated SQL:

```
select o.uri, e.eid, e.start_time, e.end_time
from events as e, eolink as eo, objects as o,
     eventclasses as ec, objectclasses as oc
where e.eid = eo.eid and eo.oid = o.oid and
      o.oclid = oc.oclid and e.eclid = ec.eclid and
      oc.name = 'Neural' and ec.name = 'hand movement'
      and e.metadata->>'whichHand' = 'left' and
      o.metadata->>'patientID' = '100'
order by e.eid, o.uri;
```

DB schema (Postgres version)



System architecture



Salient Features of the SQL Approach

Abstraction and security	•The connection to the database is via ODBC connection and stored procedures •Additional layer of security as the users do not directly interact with the database •Ensures easy updates to SQL logic with minimal change to front-end code
Indexing and retrieval	• The tables are indexed on URI • Quicker retrieval of relevant information via the query interface.
Dynamic querying	• Pipe delimiters and a decoding function makes the ingestion procedure generic • Dynamic queries are used to parse parameters from the front-end interface.
Scope for normalization	•Important tables are linked via PK-FK relationship •Mapping tables to better normalize the relationships will be introduced in the future
Cloud implementation	• Currently - SQL Server via ODBC, Azure Data Studio, and PostgreSQL • Classes on the interface offer functionality to extend usage to cloud systems for DBMS and file storage.