DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

SAGE: A Self-Optimizing Engine for Semantic Operators (Umbrella Project)

Keywords: Semantic operators, physical query optimization, learning proxy functions, statistical guarantees, vector index, declarative quality control

Problem:

1. Motivation and Vision

Large Language Models (LLMs) have enabled semantic operators—FILTER, JOIN, ORDER BY, GROUP BY, AGGREGATION—that operate over unstructured and semi-structured data using natural-language predicates. Systems such as LOTUS [1] and follow-up work have shown how to expose these operators declaratively, but their execution remains expensive, brittle, and operator-specific.

Recent work makes important progress:

- FDJ [2] reduces semantic JOIN cost by learning featurized decompositions that proxy LLM decisions with guarantees
- LLM ORDER BY [3] studies multiple physical access paths and introduces a budget-aware optimizer for ranking quality vs. cost
- Palimpzest [4] proposes physical query optimization for single-table queries only

This proposal asks a more ambitious question:

Can we build a single, self-optimizing semantic query engine that supports all semantic operators, learns reusable proxy functions across queries and datasets, minimizes LLM calls by construction, and provides declarative guarantees on recall, precision, and cost?

We propose SAGE (Semantic Adaptive Generalized Engine), a next-generation execution engine that unifies FDJ-style proxy learning, embedding-native access paths, and verifier-guided program synthesis across all semantic operators.

2. Unifying Insight: Semantic Operators as Decision Problems

At a fundamental level, all semantic operators reduce to **LLM-defined decision or scoring functions**:

Operator	Canonical semantic form
FILTER	Does tuple *t* satisfy NL predicate *p*?
JOIN	Does pair *(t_1, t_2)* satisfy NL predicate *p*?
ORDER BY	What is the relative order / score of *t* under NL criterion *p*?

GROUP BY	Do t_1 and t_2 belong to the same latent semantic group?
AGGREGATE	What summary over a semantically defined group is correct?

Key observation:
Each operator can be expressed as a function

$f_p(x) \rightarrow \{0,1\}$, R, or permutation

where p is a natural-language predicate and x is a tuple, pair, or set.

SAGE's core abstraction is to treat LLM calls as **ground truth oracles**, and to *learn cheaper executable approximations* that satisfy **user-declared quality constraints**.

3. From FDJ to a General Proxy Learning Framework

3.1 What FDJ Gets Right

FDJ introduces three crucial ideas:
- LLM calls are too expensive to use exhaustively
- Feature extraction + cheap predicates can approximate LLM decisions
- Statistical guarantees on recall/precision must be first-class

3.2 Fundamental Limitations of FDJ

However, FDJ is intentionally conservative:
- Proxy functions are CNF-structured
- Feature → label relationships are monotonic
- Proxies are operator-specific (JOIN only)
- No reuse across queries or predicates

These constraints simplify analysis but severely limit expressiveness.

4. SAGE: A Unified Semantic Operator Engine

4.1 Architecture Overview

SAGE consists of five interacting layers:

1. **Semantic IR (Intermediate Representation)**
   A logical plan with semantic operators annotated with:
   - NL predicate
   - Target operator (FILTER, JOIN, ORDER BY, …)
   - Declarative quality constraints

2. **Embedding & Representation Layer**
   - Multiple embeddings per tuple (content, attributes, structure)
   - Learned *operator-aware* embeddings (e.g., join-sensitive, order-sensitive)
3. **Proxy Function Synthesis Layer**
   - Learns executable proxy programs (not just CNF)
   - Supports non-monotonic logic, piecewise functions, trees, small neural nets
4. **Verifier & Feedback Loop**
   - Cheap verifier checks proxy outputs against sampled LLM calls
   - Guides refinement and termination
5. **Cost-Based Semantic Optimizer**
   - Chooses access paths, proxy complexity, and fallback strategies
   - Generalizes ORDER BY access-path selection to all operators

5. Beyond CNF: Flexible, Non-Monotonic Proxy Functions

Why restrict ourselves to CNF and monotonicity of FDJ?

We should not.

SAGE generalizes FDJ's featurized decompositions into **Executable Semantic Proxies (ESPs)**:

- Decision trees over features
- Small synthesized programs (Python / SQL UDFs)
- Learned classifiers over embeddings
- Hybrid symbolic–neural functions

Key idea

Instead of *assuming* monotonicity, SAGE:

- **Learns proxies under verification**
- **Rejects proxies that violate guarantees**
- **Allows non-monotonic decision boundaries**

This aligns with our intuition: **correctness comes from verification, not syntactic restriction**.

6. Verifier-Guided Program Synthesis for Semantic Operators

Why not let the LLM write a fast function and write a verifier that checks it?

SAGE adopts a CEGIS-style loop (Counterexample-Guided Inductive Synthesis):

1. LLM proposes a **candidate proxy program**
2. Engine evaluates it on sampled data
3. Verifier checks:
    - Recall ≥ target
    - Precision ≥ target
    - Cost ≤ budget
4. Counterexamples are fed back to the LLM
5. Proxy is refined until constraints are met

Termination condition:
Stop as soon as the proxy satisfies *declaratively specified constraints*.

This directly supports:

```
SEMANTIC JOIN ...
RECALL >= 0.95
PRECISION >= 0.99
COST <= $0.10
```

7. Declarative Quality & Cost Constraints

SAGE generalizes FDJ's recall/precision model and ORDER BY's budget awareness:

Users specify:

- **Recall / precision**
- **Ranking fidelity (e.g., nDCG ≥ τ)**
- **Monetary or latency budget**
- **Confidence (δ)**

Engine guarantees:

- Statistical bounds when possible
- Fallback to LLM when uncertainty is high

This makes **semantic optimization first-class**, not heuristic.

8. Embedding-Native Access Paths

We propose using ideas from **graph-based vector indexes**—this is critical.

8.1 Embeddings as Physical Data Structures

Instead of using embeddings only as *filters*:

- Pre-build **semantic neighborhood graphs** over text columns
- Maintain **operator-specific similarity spaces**

Example:

- JOIN: "Items similar to *x* tend to join with similar *y*"
- GROUP BY: Clusters correspond to latent semantic groups
- ORDER BY: Local neighborhoods preserve relative ranking

8.2 Online Predicate Challenge

NL predicates change every query

SAGE addresses this by:

- Maintaining **predicate-agnostic embeddings**
- Learning **predicate-conditioned projections**
- Reusing neighborhoods as *candidate generators*, not final answers

This parallels how **HNSW graphs** (popular vector index) are reused across queries.

9. Online Clustering & Cross-Query Knowledge Transfer

Why not cluster true/false pairs and reuse them?

SAGE introduces **Semantic Experience Memory**:

- Stores embeddings of evaluated examples
- Clusters by *reason for decision*, not just label
- Transfers knowledge across:
    - Queries
    - Datasets
    - Operators

This enables:

- Faster warm-up for new queries
- Fewer LLM calls over time
- Continual self-improvement (DSPy-like, but operator-aware)

10. Extending to ORDER BY, GROUP BY, AGGREGATION

ORDER BY

- Proxy learns *partial order* or scoring approximation
- Optimizer selects between:
  - Pointwise
  - Pairwise
  - Listwise
  - Proxy-based ranking
    based on budget and required fidelity

GROUP BY

- Treated as **semantic clustering**
- Proxy predicts cluster IDs
- LLM only verifies boundary cases

AGGREGATION

- Apply aggregation on proxy-generated groups
- Sample-and-verify summaries

All operators share:

- Proxy learning
- Verifier logic
- Embedding infrastructure

11. Expected Contributions

- **A unified semantic operator execution engine**
- **Verifier-guided proxy synthesis beyond monotonic CNF**
- **Declarative recall–precision–cost optimization**
- **Embedding-native access paths for semantic operators**
- **Cross-query and cross-operator learning**

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

12. Impact

SAGE moves semantic data processing from:

"LLMs everywhere, heuristics everywhere"
to
"LLMs as oracles, databases as optimizers"

This directly advances:

- LLM-powered DBMSs
- Cost-efficient analytics
- Long-running semantic workloads
- Research at the intersection of DB, ML, and programming languages

FDJ [2] showed **how to reduce cost for one operator under strong assumptions**.
LLM ORDER BY [3] showed **how access paths matter but lack unification**.

SAGE generalizes both into a **self-optimizing semantic execution engine** where:

- Proxies are learned, verified, and reused
- Embeddings are first-class physical structures
- Quality and cost are declarative
- LLM calls are the *exception*, not the rule

[1] Semantic Operators: A Declarative Model for Rich, AI-based Data Processing
[2] Featurized-Decomposition Join: Low-Cost Semantic Joins with Guarantees
[3] Access Paths for Efficient Ordering with Large Language Models
[4] A Declarative System for Optimizing AI Workloads

Project: In this project, the student(s) will 1) get to know about the physical query optimization for semantic logical plans and related work [1, 2, 3, 4], 2) evaluate SOTA open-source baselines, 3) identify the feasible research direction among the ones mentioned above based on the student's interest, background, and evaluation results, 4) implement specific ideas and compare the performance with the baselines. Students will analyze baseline's success and failure cases, as well as our ideas, ultimately towards building a component inside SAGE.

Plan:

1. Get to know the physical query optimization for semantic logical plans and understand key related work.
2. Select key SOTA open-source baselines. Evaluate and analyze their performance (accuracy and efficiency) including the case analysis (failures and successes). Both quantitative and qualitative analyses are required. Identify their bottlenecks and set these as the research challenges.
3. Propose techniques to resolve such challenges. Our first direction may be Sections 8-9 above, leveraging vector embeddings and clustering to replace expensive LLM calls as a proxy.
4. Write a report or hopefully a paper.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Supervisor:	Prof. Anastasia Ailamaki, anastasia.ailamaki@epfl.ch
Responsible collaborator(s):	Kyoungmin Kim, kyoung-min.kim@epfl.ch
Duration:	3-6 months