CADENZA in Action: Breaking the Monolith with Intent-Dependent Plan Spaces for Semantic Queries

Jaehyun Ha
jhha@dblab.postech.ac.kr
GSAI, POSTECH
Pohang, Korea

Yongjoo Park
yongjoo@illinois.edu
Univ. of Illinois Urbana-Champaign
Urbana, USA

Wook-Shin Han*
wshan@dblab.postech.ac.kr
GSAI, POSTECH
Pohang, Korea

ABSTRACT

Semantic query processing engines execute semantic operators, whose behavior is specified by natural-language intents, via model inference over multimodal data. Most existing optimizers optimize the operators at the granularity of monolithic implementations—such as LLMs and embedding models—forcing a trade-off between expensive model calls and cheaper alternatives that fail to capture intent-dependent semantics. We present CADENZA, a semantic operator optimizer that compiles an intent into decomposed steps, selects concrete physical implementations for each step, and tunes their parameters under user-specified quality–latency–cost preferences. In this demonstration, users interact with CADENZA through a web interface over multimodal databases, exploring how an intent is decomposed into alternative plans, how each plan is optimized, and how different preferences yield different winning plans.

PVLDB Reference Format:
Jaehyun Ha, Yongjoo Park, and Wook-Shin Han. CADENZA in Action: Breaking the Monolith with Intent-Dependent Plan Spaces for Semantic Queries. PVLDB, 19(12): 4634 - 4637, 2026.
doi:10.14778/3827998.3828084

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/postechdblab/CADENZA.

1 INTRODUCTION

With the advent of Large Language Models (LLMs), a new class of *semantic query processing engines* (SQPEs) has emerged [4, 5, 8, 10, 11]. Semantic operators—such as `SemFilter`, `SemJoin`, and `SemMap`—are operators whose behaviors are specified by natural-language task descriptions (i.e., intents) and executed via model inference over multimodal data (e.g., text and images). These operators simplify complex multimodal analysis. For example, suppose an e-commerce platform onboards a vendor who provides product descriptions (text with brand, category, specs, etc.) and product photos (images with logos but no metadata) as two relations with no shared identifier. To build a unified catalog, a data engineer can execute a `SemJoin` to *"match product descriptions with product photos of the same brand."*

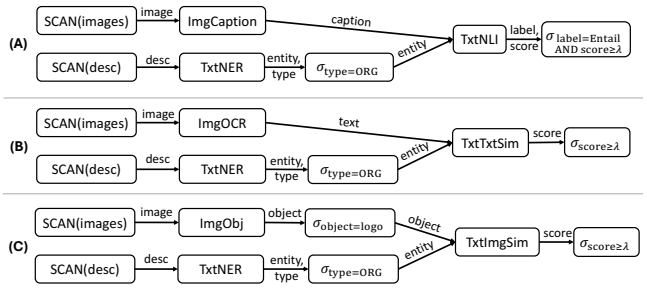


Figure 1: Three candidate logical plans for `SemJoin` with *"match product descriptions with product photos of the same brand."* Each node is an operator and each edge denotes the attribute consumed by the next operator.

Most existing semantic query optimizers [4, 5, 8–11] optimize at the granularity of a monolithic operator implementation—e.g., choosing an LLM, an embedding model, or a cascade—rather than exploring alternative decompositions of the operator's task. A large LLM can ensure high accuracy yet incurs substantial cost and latency; a smaller LLM or embedding model reduces cost, but embeddings are often too coarse to capture the nuances of arbitrary intents. For instance, embedding an entire product description into a single vector dilutes fine-grained brand signals among other attributes, degrading match quality regardless of the model chosen. The root cause is that these systems never look inside the intent: they cannot decompose it into finer-grained, specialized steps that would admit both cheaper and more accurate execution.

Our key insight is that an intent can be compiled into a small logical DAG where each node is a specialized model or a control-flow operator, dynamically selected or synthesized in a query-aware manner. This enables both cheaper and more accurate execution than monolithic approaches. For example, the brand-matching query above can be translated into any of a few candidate plans (Figure 1). While each plan extracts brand entities via NER, they employ different visual processing strategies: Plan A generates captions (e.g., via BLIP [3]) and verifies brand entailment via NLI (e.g., via DeBERTa [1])—cheap, but accuracy depends on whether the caption mentions the brand; Plan B reads printed text via OCR for direct text comparison—accurate when brand text is visible, but fails otherwise; and Plan C detects logos (e.g., via YOLO [7]) and matches them via text–image similarity (e.g., CLIP [6])—the most robust but also the most expensive. The optimal plan is chosen based on the user's preferences over quality, latency, and cost.

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828084

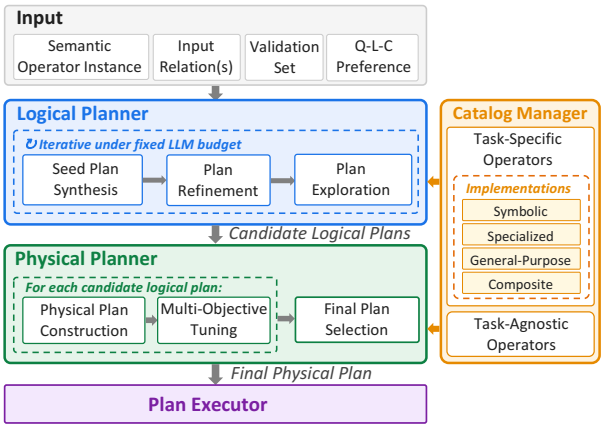


Figure 2: System architecture of CADENZA.

We present CADENZA, a semantic operator optimizer that automatically discovers such alternative decompositions, selects concrete implementations for each decomposed step, tunes their parameters under multi-objective optimization, and returns the best plan under the user's quality–latency–cost preference. In this demonstration, users interact with CADENZA through an interactive web interface over multimodal databases, exploring how an intent is decomposed into alternative plans, how each plan is optimized, and how different preferences yield different winning plans.

2 SYSTEM OVERVIEW

This section presents an overview of CADENZA, a standalone optimizer with a built-in plan executor. Integration with existing SQPEs is supported through lightweight dispatch hooks and adapter patterns. We describe the system's inputs and operator catalog, then present the logical planner that decomposes intents into logical plans, and the physical planner that compiles, tunes, and selects the final plan under user-specified quality–latency–cost preferences (Figure 2).

Input. CADENZA takes as input (i) a *semantic operator* (e.g., `SemJoin`) and its *intent*, (ii) the multimodal input relation(s), (iii) an optional *validation set*, and (iv) a quality–latency–cost (Q-L-C) *preference*. Formally, an *intent* is a natural-language specification of a semantic operator instance's desired behavior, with no fixed algebraic form; the same intent therefore admits multiple valid decompositions that CADENZA surfaces as alternative logical plans. Preferences are weight vectors (w_q, w_l, w_c), with preset profiles *Quality First*, *Balanced*, and *Budget*. Following prior work [5, 8], we estimate each candidate plan's quality, latency, and cost on a small *proxy-labeled* validation set. If none is provided, CADENZA labels a subset with an oracle model (e.g., GPT-4), reused across all candidates. Because proxy labels are used only to *rank* candidate plans rather than to define ground truth, the ranking stays stable under moderate labeling noise; users may also supply their own validation set. Note that U scores only the selected plan's deployment cost; the (plan-dependent) optimization overhead is reported separately in the trial log and amortized over deployment rather than charged to U.

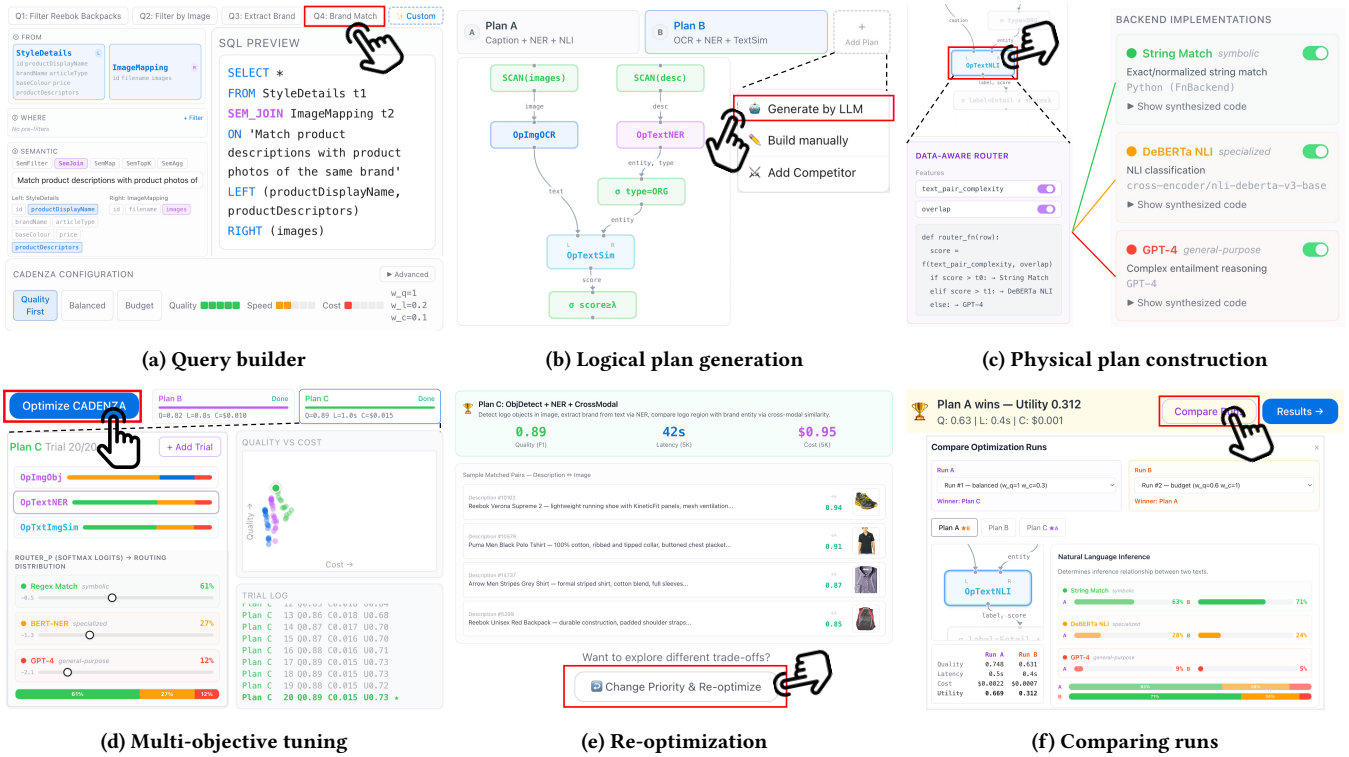
Catalog Manager. The catalog manager maintains the logical operators and their physical implementations. Operators fall into two classes. *Task-specific operators* are typed, per-tuple inference steps that each correspond to a well-defined task formulation in multimodal model ecosystems (e.g., Hugging Face), with a typed signature and a stable output schema independent of backend. Each runs inference on input attributes and appends produced columns (e.g., ImgCap appends a *Caption* column). We group task-specific operators by the database role of their outputs into five categories: *structural* (indexes substructures, e.g., splitting, region detection), *attributive* (unary descriptors, e.g., classification, NER), *associative* (relations over elements or pairs, e.g., NLI, pair scoring), *generative* (derived artifacts as new semantic objects, e.g., summarization, captioning), and *latent* (vectors for similarity-based retrieval). *Task-agnostic operators*—standard relational operators such as filter (σ), join ($\bowtie$), and aggregation (γ)—connect task-specific steps within a plan. The full catalog, spanning text and image modalities, is browsable in the demo interface; new operators are added by registering a task signature and assigning a category.

On the physical side, each operator has one or more *implementations*, organized by backend family: *symbolic* (e.g., string matching), *specialized* (e.g., DeBERTa-NLI, CLIP), *general-purpose* (e.g., GPT-4), and *composite* (e.g., multi-stage pipelines). For the supported operators, we precompile implementations offline, except symbolic ones that require query-specific knowledge and are instantiated on demand. Precompiled implementations are maintained via an agentic pipeline that automatically discovers and integrates models from Hugging Face, keeping the catalog up to date. Task-agnostic operators use standard implementations.

Logical Planner. The logical planner determines *what* to compute by decomposing the intent into a DAG of operators—a *logical plan*. Each plan interleaves task-specific operators with task-agnostic operators that wire intermediate results together. Because a single intent admits multiple valid decompositions, the planner first synthesizes a small set of *seed plans* under a fixed LLM budget: it prompts an LLM with the intent, schemas, catalog, and few-shot examples to propose a decomposition. Each draft is verified against schema and dataflow constraints, and invalid drafts are refined (*plan refinement*). The planner then expands the seed set via rule-based transformations (*plan exploration*) without additional LLM calls, producing structurally diverse candidate plans.

Figure 1 shows three candidates for our running example; all extract brand entities from descriptions via NER, but differ in how they process images. Plan A (*NER+Caption+NLI*) captions each image, then takes the cross product of captions and extracted brand entities, scores entailment via NLI, and filters on the score ($\sigma_{score \geq \lambda}$). Plan B (*NER+OCR+TextSim*) reads text printed on the product via OCR and compares it to the brand entity via text similarity. Plan C (*NER+ImgObj+TxtImgSim*) detects objects in images, filters for logo regions ($\sigma_{object=logo}$), scores each logo–brand pair via CLIP-based text–image similarity, and filters on the score ($\sigma_{score \geq \lambda}$).

Physical Planner. The physical planner determines *how* to execute each logical plan by binding every operator to concrete implementations with tunable parameters. Task-agnostic operators such as $\sigma_{sim \geq \lambda}$ become standard filters with a tunable threshold λ. Each



(a) Query builder (b) Logical plan generation (c) Physical plan construction

(d) Multi-objective tuning (e) Re-optimization (f) Comparing runs

Figure 3: Demo walkthrough. S1 (a–d): Under Balanced preference, a SemJoin query is decomposed into three plans; Plan C wins. S2 (e–f): Under Budget preference, re-optimization yields a different winner, Plan A.

task-specific operator is compiled into a *routed ensemble*: the planner retrieves candidate backends from the catalog and synthesizes a *data-aware router* by prompting an LLM with the operator specification and the ordered backend set. The LLM generates (i) lightweight feature-extraction code that computes cheap, non-inferential signals (e.g., text length, keyword overlap) and (ii) scoring code that maps these features to a difficulty score $s \in [0, 1]$. A fixed bucketing template then uses s and $N-1$ tunable cutpoints to dispatch each tuple to one of N backends. For instance, the TxtNLI operator in Plan A may route pairs whose caption already contains the brand name to a symbolic string matcher, while ambiguous pairs are forwarded to DeBERTa-NLI or GPT-4. The routing thresholds and per-backend parameters together form the tunable space Θ.

For each candidate plan, CADENZA tunes Θ via Bayesian optimization on the validation set. Each trial executes the physical plan on the validation set and measures quality Q (e.g., F1), latency L, and cost C (API plus local GPU cost). By default, these are combined into a scalar utility $U = w_q Q - w_l \hat{L} - w_c \hat{C}$, where $Q \in [0, 1]$ and $\hat{L}, \hat{C}$ are latency and cost *log-normalized* to $[0, 1]$. The weights $(w_q, w_l, w_c) \geq 0$ thus encode *relative* importance rather than trading raw seconds against dollars, and users set them via the preset profiles or a slider. Other formulations—such as maximizing one metric subject to hard constraints on the others—are also supported, as the tuning loop is agnostic to the choice of objective function. The best plan is then selected for execution. Crucially, the Q-L-C preference steers not only routing parameters but potentially the winning *logical plan* itself: a *Quality First* setting may favor Plan C's targeted logo matching, whereas a *Budget* setting may prefer Plan A's lighter captioning strategy.

3 DEMONSTRATION: TWO SCENARIOS

We demonstrate CADENZA's end-to-end workflow—from query construction through plan decomposition, physical compilation, and multi-objective tuning—on the E-Commerce scenario from SemBench [2][1]. The user plays a data engineer who must match product descriptions with product photos of the same brand via `SemJoin`. During the demo, the user explores how this intent yields different logical plans, how each is compiled and tuned, and how changing preferences selects a different winner.

Beyond the curated scenarios, the interface supports custom dataset registration and flexible plan construction via a drag-and-drop canvas. Users can assemble logical plans from 36 operators—22 for text and 14 for images—while real-time schema validation ensures logical correctness. For example, a user can register a new image collection and assemble a `SemFilter` for *"images that display a given phrase"* on the canvas—chaining OCR → string match—exercising a different operator and modality from the SemJoin scenario without writing code.

[1]We preload the SF=100 subset (100 descriptions and 100 images) with cached model weights.

3.1 S1: One Intent, Three Strategies

(a) Query Builder. In our first scenario, the user opens the query builder (Figure 3a), selects the E-Commerce dataset and picks preset Q4, a `SemJoin` that binds two input relations: product descriptions (text) and product images. The preference panel is set to *Balanced*, and the validation set size and tuning budget are configured through advanced settings.

(b) Logical Plan Generation. Clicking *"Generate by LLM"* triggers the logical planner, which synthesizes three candidate plans (Figure 3b), each a distinct plan for the same intent (see also Figure 1). By clicking *"Add Competitor,"* the user can also add monolithic baselines for comparison—plans that execute the `SemJoin` without decomposition. For this walkthrough, the user adds two baselines; their physical implementations are chosen in the next step.

(c) Physical Plan Construction. The user configures a physical implementation for each plan (Figure 3c). Selecting Plan A reveals each task-specific operator's routed ensemble. For example, `TxtNLI` offers a symbolic string matcher, DeBERTa-NLI, and GPT-4, with synthesized code expandable on click. Users can toggle backends and observe routing shifts in real time; unconfigured operators include all backends by default. For the baselines, the user selects single-LLM and embedding-cascade implementations of `SemJoin` for comparison with the decomposed plans.

(d) Multi-Objective Tuning. The user clicks *"Optimize CADENZA"* to launch Bayesian optimization, which tunes each decomposed plan's routing thresholds and backend parameters on the validation set (Figure 3d). A live dashboard tracks progress through a trial log, a Pareto scatter, and per-operator routing cards. Under Balanced preferences, Plan C's fine-grained decomposition lets the router assign easy inputs to cheap backends while reserving LLM calls for hard cases. The user can also drag a routing-ratio slider to set backend proportions and fire a trial, observing how the metric triple responds. The user then clicks *"Optimize Baselines"* to tune the two monolithic plans under the same setting; their results join the same Pareto chart. Table 1 reports the measured numbers.

Table 1: Measured Q, L, and C for the running example.

Plan	$Q\uparrow$	L (s)$\downarrow$	C (\$)$\downarrow$
CADENZA (ours)	**0.82**	**566**	**0.80**
Single LLM	0.76	25,323	2.59
Embedding Cascade	0.63	4,208	1.20

3.2 S2: Preference Changes Plan

(e) Re-optimization. After tuning completes, the results screen displays Plan C's quality, latency, and cost alongside sample matched pairs. While Plan C delivers the best quality under Balanced preferences, its cost may be too high for production at scale. The user clicks *"Change Priority & Re-optimize"* (Figure 3e), switching to *Budget* to re-optimize with a cost-conscious objective.

(f) Comparing Runs. The user clicks *"Compare"* to view the Balanced (Plan C wins) and Budget (Plan A wins) runs side by side (Figure 3f). Per-operator routing shows the Budget objective shifting work away from expensive backends. Plan A (NER+Caption+NLI) now wins through cheap captioning rather than object detection. Thus, preferences steer both routing and the *logical plan* selected for the same intent.

4 CONCLUSION

We demonstrated `CADENZA`, a semantic operator optimizer that decomposes an intent into alternative logical plans, compiles each into a physical plan with routed ensembles of heterogeneous backends, and tunes them under user-specified quality–latency–cost preferences. Through an interactive demo, users can explore how an intent is decomposed, how each plan is optimized, and how changing preferences yields a different winning plan—making intent-dependent semantic query optimization transparent and practical.

ACKNOWLEDGMENTS

This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2025-00517736, 95%) and the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00509258, Global AI Frontier Lab, 5%).

REFERENCES

[1] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-Enhanced BERT with Disentangled Attention. In *International Conference on Learning Representations*. https://openreview.net/forum?id=XPZIaotutsD

[2] Jiale Lao, Andreas Zimmerer, Olga Ovcharenko, Tianji Cong, Matthew Russo, Gerardo Vitagliano, Michael Cochez, Fatma Özcan, Gautam Gupta, Thibaud Hottelier, et al. 2025. SemBench: A Benchmark for Semantic Query Processing Engines. *arXiv preprint arXiv:2511.01716* (2025).

[3] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*. PMLR, 12888–12900.

[4] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. 2025. Palimpzest: Optimizing ai-powered analytics with declarative query processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*. 2.

[5] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4171–4184.

[6] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PmLR, 8748–8763.

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[8] Matthew Russo, Sivaprasad Sudhir, Gerardo Vitagliano, Chunwei Liu, Tim Kraska, Samuel Madden, and Michael Cafarella. 2025. Abacus: A Cost-Based Optimizer for Semantic Operator Systems. *arXiv preprint arXiv:2505.14661* (2025).

[9] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G Parameswaran, and Eugene Wu. 2024. Docetl: Agentic query rewriting and evaluation for complex document processing. *arXiv preprint arXiv:2410.12189* (2024).

[10] Jiayi Wang and Jianhua Feng. 2025. Unify: An unstructured data analytics system. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 4662–4674.

[11] Jiayi Wang and Guoliang Li. 2025. Aop: Automated and interactive llm pipeline orchestration for answering complex queries. In *Proceedings of the Conference on Innovative Database Research (CIDR)*.