MM-mapsearch: Workload-Aware Mapping Selection

Pavel Koupil
Charles University
Prague, Czech Republic
pavel.koupil@matfyz.cuni.cz

Bedřich Mazourek
Charles University
Prague, Czech Republic
mazourek420@gmail.com

Jáchym Bártík
Charles University
Prague, Czech Republic
jachym.bartik@matfyz.cuni.cz

Irena Holubová
Charles University
Prague, Czech Republic
irena.holubova@matfyz.cuni.cz

Abstract

Modern database systems offer several alternative representations of the same data, both within one model and across multiple models. These choices have a strong influence on performance. But manually identifying an efficient mapping is challenging, especially in multi-model environments that combine heterogeneous operators.

In this paper, we present *MM-mapsearch*, an automatic advisor for selecting workload-aware multi-model mappings. A simplified categorical schema defines the search space and enables uniform reasoning about structural alternatives. *MM-mapsearch* analyses a weighted workload, explores feasible schema variants using Monte Carlo Tree Search, and estimates their performance with a plan-structured neural predictor that generalizes across relational, document, and graph systems.

Keywords

Multi-model DBMS, workload-aware optimization, data migration

1 Introduction

Modern database management systems (DBMSs) usually allow several ways to represent the same data. Even within one data model, the data can be organized using different structural designs that vary in their level of nesting, granularity, or normalization. Prior work has shown that such representation choices can lead to substantial performance differences across workloads [6]. Many systems also support multiple data models[1] simultaneously, which enables the combination of representations across models [13]. While all these options provide flexibility, they also introduce significant performance differences.

Choosing an appropriate mapping is, therefore, a challenging task. Multi-model workloads may contain heterogeneous access patterns and operator families, and each representation affects operator behavior, indexing, and traversal differently. As the number of possible mappings grows rapidly, manual evaluation becomes time-consuming, error-prone, and often unrealistic without a deep understanding of the underlying systems.

Schema evolution and workload-driven optimization have been studied across various DBMSs. For multi-model databases, recent work shows that deep reinforcement learning can tune system parameters [17, 18], but these approaches do not modify the schema or the underlying data representation. In NoSQL systems, schema evolution has been explored extensively: [16] surveys practical challenges of schema evolution and migration, MigCast [5] estimates the cost and performance implications of data-model changes, [3] introduces a formal taxonomy of schema modifications, [2] proposes generic evolution techniques applicable across both NoSQL and relational models, [4] presents a categorical framework for migrations between NoSQL models, Darwin [15] offers a platform for multi-model data evolution, and [7] defines basic operations for both evolution and migration of multi-model data. Although these works highlight the importance of structural adaptation, none address *query-driven* schema changes in multi-model settings.

The natural solution – automatically suggesting or adapting data representations in multi-model settings – introduces several challenges. (1) The search space of possible mappings grows exponentially with the number of entities and their possible embeddings, inlinings, or placements across models. (2) Multi-model query workloads may combine different operator families (relational, document, graph), and the impact of a representation change often propagates across model boundaries. (3) Evaluating each candidate mapping requires estimating performance for heterogeneous operators, which is difficult without a unified model of query-plan behavior.

To address these challenges, we introduce *MM-mapsearch*, a tool that analyses multi-model workloads and recommends performance-oriented schema or mapping changes. Its main contributions are:

- A simplified categorical layer [10] that provides a uniform representation of relational, document, and graph structures and enables principled cross-model reorganization.
- Automatic analysis of weighted multi-model workloads, with schema and mapping changes derived directly from query patterns, removing the need for manual inspection.
- Systematic exploration of model assignments and embedding variants via Monte Carlo Tree Search (MCTS), with a reward function balancing predicted latency and migration effort
- Performance estimation for each candidate representation using an extended plan-structured neural predictor [14], steering the search toward high-quality schema variants.
- A full-featured prototype implementation integrating workload analysis, schema-variant generation, neural performance prediction, and an interactive interface for inspecting and comparing recommended mappings.

2 Proposed Approach

The goal of *MM-mapsearch* is to identify an optimal representation across logical models – relational, document, and graph – for

[1] https://db-engines.com/en/ranking

EDBT '26, Tampere (Finland)

10.48786/edbt.2026.68

a given workload of queries. It searches for mappings that minimize query latency and migration cost by exploring alternative schema variants and predicting their performance. Fig. 1, 2, and 3 shows the overall workflow, composed of three main stages:

(1) **Categorical representation**: The schema is converted into a simplified categorical form capturing complex properties and their embedding or reference links, which together with the weighted workload define the search space.
(2) **MCTS space exploration**: The algorithm explores model assignments and embedding variants, maintaining a directed acyclic graph (DAG) of visited states and using a reward function balancing predicted latency and migration effort.
(3) **Performance prediction**: A trained neural model estimates query latency for each candidate mapping based on system-specific plans represented in a unified operator-level form.

Example 2.1. Consider a simplified TPC-H[2] scenario in Fig. 1, where order data are stored in a relational DBMS (purple), while information about parts and suppliers resides in a graph DBMS (blue). The workload contains three queries (Q_1, Q_2, Q_3) with associated execution frequencies (w_1, w_2, w_3). The goal is to find a representation that minimizes overall latency, whether by keeping the current placement, migrating dataset(s), or reorganizing all data under a single model.

MM-mapsearch evaluates the alternatives automatically. The schema and workload in Fig. 1 are encoded into the root state of the MCTS search: each complex property is assigned a model and an optional embedding parent, represented by two vectors that form the initial configuration. MCTS (Fig. 2) then explores neighboring states, where each edge corresponds to a single change in model assignment or embedding. During search, the four MCTS phases are applied repeatedly: a promising state is selected, one modification is expanded, its workload latency is estimated using the neural predictor (Fig. 3), and the resulting average value, total reward, and visit counts are backpropagated along the path to the root. The final output is a recommended mapping together with predicted performance gains.

Contrary to the example, the initial mapping doesn't have to be multi-model. *MM-mapsearch* can also start from a single-model schema and explore its multi-model alternatives. The resulting mapping can then directly replace the original representation, keeping the same query interface while improving performance.

2.1 Categorical Multi-Model Representation

The input to the framework is a categorical schema $\mathcal{S} = (O, M, \circ, 1)$, where O is the set of objects representing data properties, M is the set of morphisms describing connections between properties, $\circ$ denotes morphism composition, and 1 is the identity morphism. In the general form [10], morphisms capture a rich set of structural relationships, such as *"has an identifier"*, *"has a property"*, *"has a role"*, or *"is a"*. The categorical schema $\mathcal{S}$ can be created manually [12] or inferred automatically from sample data instances [11].

For the purpose of mapping optimization *MM-mapsearch* operates on a simplified variant of $\mathcal{S}$ that retains only *complex properties* and their *embedding* or *reference* morphisms. Complex properties correspond to entire structural units such as tuples, (nested) documents, graph vertices, or edges (e.g., *Orders* or *Supplier* in Fig. 1).

The resulting schema can be viewed as a directed graph $\mathcal{G} = (V, E)$ induced by the objects and morphisms of $\mathcal{S}$, where vertices $v \in V$ represent complex properties and edges $e = (v_i, v_j) \in E$ denote embedding, inlining, or reference links. Together with this simplified schema, the user provides a workload $Q = \{(q_i, w_i)\}$ of queries q_i and their execution frequencies w_i. Additional constraints may be attached to edges to specify whether embedding, inlining, or referencing is permitted or mandatory, thus defining the feasible search space to be explored.

[2] https://www.tpc.org/tpch/

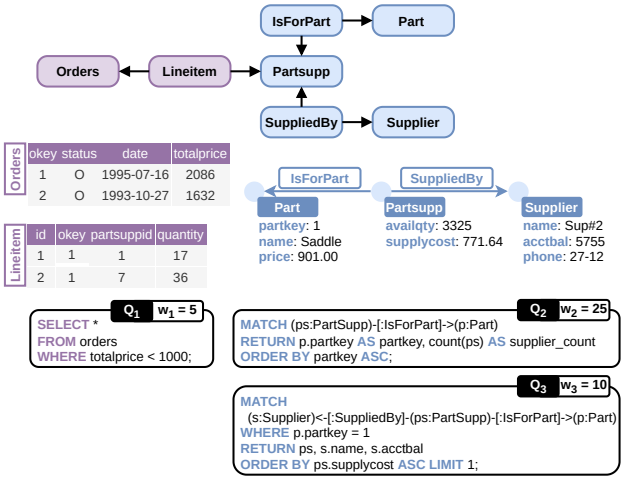

Figure 1: Categorical input representation of the schema and workload.

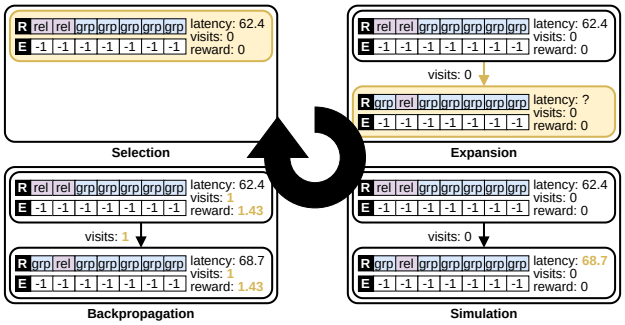

Figure 2: Exploration of the mapping space using MCTS.

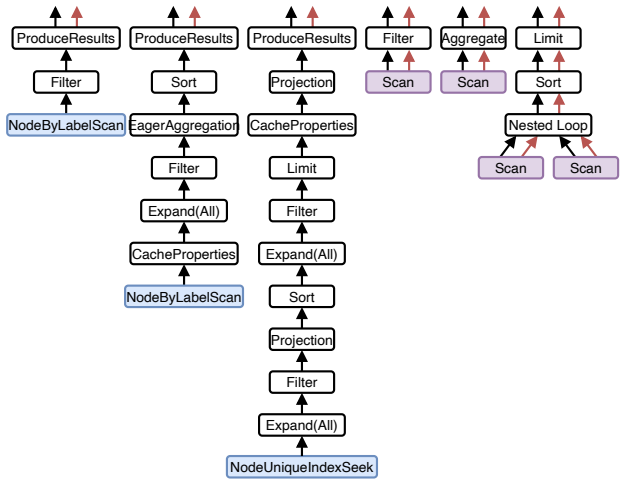

Figure 3: Plan-structured neural predictor for estimating query performance.

2.2 Mapping-Space Exploration using MCTS

Given the simplified schema and workload, *MM-mapsearch* employs MCTS to explore possible model assignments and embedding configurations that jointly determine the multi-model mapping. Each *state* $S = (R, E)$ consists of two vectors: $R[i] \in \{\text{rel}, \text{doc}, \text{grp}\}$ encodes the logical model of the i-th complex property, and $E[i] \in \{-1, 0, \ldots, n-1\}$ specifies its embedding parent, with $E[i] = -1$ indicating a top-level property. *Actions* modify the state by changing a model assignment or altering an embedding relation. Each node in the search graph maintains its *visit count*, *total reward*, and *average reward*, while edges store their own visit counts. This enables a UCT2-style [1] adaptation of the *upper confidence bound* (UCB) for DAGs, ensuring consistent exploration even when different action sequences lead to the same decomposition. Before the search begins, the predicted latency of the root state is computed to establish the initial baseline. The reward is defined as $r = \frac{100}{t_m + t_e + 0.001}$, where t_m is the migration cost and t_e is the weighted query latency (both in seconds).

The MCTS iterates through four phases. During *selection*, the algorithm traverses the search space from the root, choosing at each step the child that maximizes a UCB score. Exploitation uses a node's average reward, whereas exploration relies on the visit count of the connecting edge. Selection stops once a state with untried actions is reached. In *expansion*, one such action is applied, usually modifying a single component of (R, E) to create a new mapping candidate. During *simulation*, the neural predictor estimates the workload latency for this candidate, which is converted into a reward. In *backpropagation*, every node and edge along the selected path updates its visit count and reward value.

Unlike standard tree-based MCTS, our implementation builds a DAG of visited states because many action sequences produce identical mappings. This avoids redundancy and significantly reduces the search space. To prevent cycles (e.g., repeatedly migrating a property back and forth), visited paths are recorded and actions leading to previously seen states are excluded.

Example 2.2. Figure 2 illustrates a single MCTS iteration. The root mapping is selected, and a new variant is generated by modifying one model assignment (here, $R[0] = \text{rel}$). The neural predictor estimates its workload latency (68.7 ms), producing a reward of 1.43, which is then backpropagated to update statistics along the visited path. Iterations continue until the time budget is reached, at which point the best mapping found is recommended.

2.3 Plan-Structured Neural Predictor

During the *simulation* phase, MCTS invokes a learned performance predictor that estimates workload latency for the current mapping. The model extends the *plan-structured neural network* of [14] to support multiple logical models. As illustrated in Fig. 3, the predictor is built from *operator-level neural units*, each specialized for a single operator type (e.g., `Projection`, `Selection`, `Join`). These units are dynamically assembled into a network mirroring the query plan. A unit receives an operator's *feature vector* (red arrow) and a *data vector* (black arrow) from its children and outputs (i) a latency contribution and (ii) a new data vector summarizing intermediate statistics. Computation proceeds bottom-up: leaf units (e.g., `Scan`, `NodeByLabelScan`) use only operator features, internal units concatenate their children's outputs with their own features, and the root (`ProduceResults`) emits the final latency estimate. The left three trees (blue leaves) correspond to Neo4j operators (e.g., `NodeByLabelScan`, `Expand(All)`), while the right three trees (purple leaves) represent PostgreSQL plans (e.g., `Scan`, `Filter`).

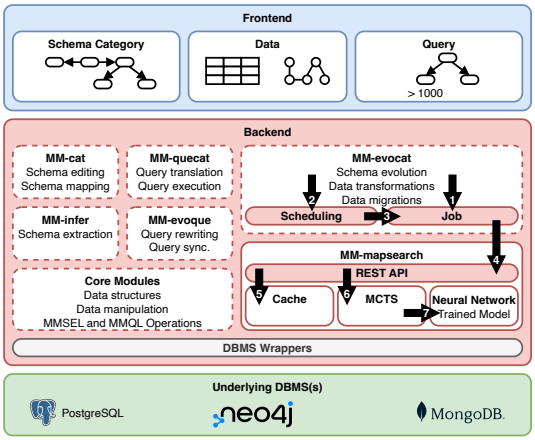

Figure 4: Architecture.

To support multi-model workloads, equivalent operator families across PostgreSQL, MongoDB, and Neo4j are mapped to unified operator types inspired by MMQL translation [9]. The feature extractors normalize numeric parameters (e.g., `Plan Rows`, `Total Cost`), convert categorical fields into vectors, and represent yes/no properties as binary flags. PostgreSQL features come from `EXPLAIN ANALYZE`; MongoDB features from JSON plans; and Neo4j features from `PROFILE`. As Neo4j lacks per-operator timings, internal nodes emit only data vectors, while the root predicts total latency.

Training data are obtained by executing benchmark workloads and recording plans and runtimes. For systems with operator-level timings, the loss aggregates errors across all plan nodes; for Neo4j, it compares only the total latency. The resulting unified model serves as a fast cost oracle, enabling MCTS to evaluate mapping alternatives without executing the workload physically.

3 Architecture and Implementation

MM-mapsearch[3] is implemented as a standalone microservice within the *MM-evocat* toolset [8] (see Fig. 4), a broader ecosystem for modeling, querying, evolving, and propagating changes in multi-model data. The toolset follows a modular, service-oriented architecture in which components communicate through REST APIs and exchange data using a unified categorical representation. The *MM-mapsearch* service is written in Python and exposes an endpoint that accepts the simplified schema category (Section 2), mapping constraints, and a weighted query workload. Based on this input, the service produces a mapping recommendation together with estimated performance characteristics. Due to its declarative interface and reliance on the common representation layer, the advisor can be easily extended to support new logical models, workload types, or integration scenarios within the *MM-evocat* ecosystem.

The microservices cooperate through a request–response workflow. Mapping analysis can be initiated either on demand (1) or through the scheduling component (2), which periodically triggers advisory jobs (3). A job instance communicates with *MM-mapsearch* via its REST API, submitting a request for a new recommendation (4). Before launching a computation, the service consults an internal cache to reuse previously generated results

[3] https://github.com/mmcatdb/aimm

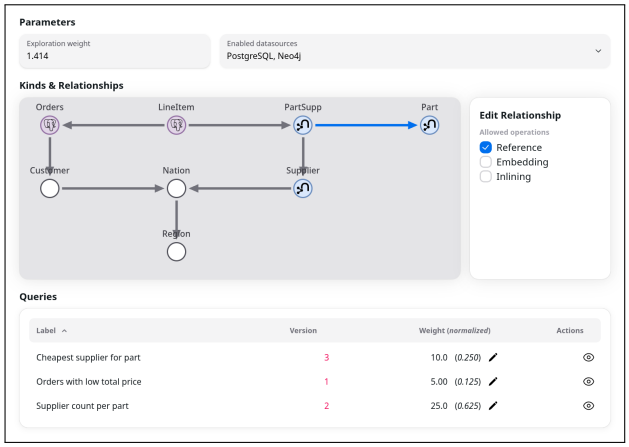

Figure 5: Input and workload selection.

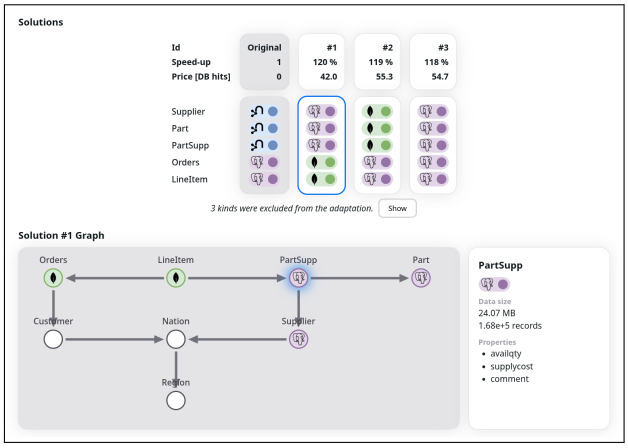

Figure 6: Mapping results and statistics.

(5), ensuring fast responses when the schema and workload have not changed. Otherwise, *MM-mapsearch* performs a fresh optimization (6-7) and returns the resulting mapping plan to the invoking component, where it is visualized and may be further propagated to schema-management or migration pipelines.

Internally, the service organizes optimization tasks as isolated Python processes. At most one optimization job is executed for a project at a time, preventing conflicting updates or excessive load. The trained neural model used for performance estimation is trained on query plans from PostgreSQL, MongoDB, and Neo4j using the TPC-H benchmark.

4 Demonstration Outline

Our interactive demo presents the complete *MM-mapsearch* workflow. Users can start with curated examples (e.g., TPC-H), which include predefined schemas and workloads, or import their own data into PostgreSQL, Neo4j, and/or MongoDB, and define custom properties and queries. Through the interface (Fig. 5), they select optimization targets, set mapping constraints (embedding, inlining, reference), assign workload weights, and specify a time budget for the search.

During optimization, the system provides continuous interactive feedback (Fig. 6). Users can track MCTS progress, inspect the current top candidates, and see how each proposed representation differs from the original. Graphical summaries highlight estimated latency gains, structural reorganization steps, and migration implications.

After the search terminates, or at any time when the user accepts a suggested mapping, the interface enables direct execution or scheduling of data migration. Up to three candidates can be inspected side by side, with detailed statistics that clarify their respective trade-offs. The demo highlights the interactivity and transparency of *MM-mapsearch*, offering a concise yet hands-on exploration of workload-aware multi-model mapping selection.

Acknowledgments

Supported by the GAČR grant no. 23-07781S and SVV grant no. 260821.

References

[1] Benjamin E. Childs, James H. Brodeur, and Levente Kocsis. 2008. Transpositions and Move Groups in Monte Carlo Tree Search. In *Proc. of IEEE CIG '09, Perth, Australia, 15-18 December, 2008*. IEEE, 389–395.

[2] Alberto Hernández Chillón, Meike Klettke, Diego Sevilla Ruiz, and Jesús García Molina. 2024. A Generic Schema Evolution Approach for NoSQL and Relational Databases. *IEEE Trans. on Knowl. and Data Eng.* 36, 7 (July 2024), 2774–2789.

[3] Alberto Hernández Chillón, Meike Klettke, Diego Sevilla Ruiz, and Jesús García Molina. 2022. A Taxonomy of Schema Changes for NoSQL Databases. *CoRR* abs/2205.11660 (2022).

[4] Annabelle Gillet and Éric Leclercq. 2025. A Categorical Representation of Multi-model Data to Prevent Data Migration Mismatch. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems LIX: Special Issue on Data Management-Principles, Technologies, and Applications*. Springer, 61–93.

[5] Andrea Hillenbrand, Maksym Levchenko, Uta Störl, Stefanie Scherzinger, and Meike Klettke. 2019. MigCast: Putting a Price Tag on Data Model Evolution in NoSQL Data Stores. In *Proc. of SIGMOD '19*. ACM, New York, NY, USA, 1925–1928.

[6] Stratos Idreos, Kostas Zoumpatianos, Brian Hentschel, Michael S. Kester, and Demi Guo. 2018. The Data Calculator: Data Structure Design and Cost Synthesis from First Principles and Learned Cost Models. In *Proc. of SIGMOD '18*. ACM, New York, NY, USA, 535–550.

[7] Pavel Koupil, Jáchym Bártík, and Irena Holubová. 2025. Model-Agnostic Evolution Management. In *Research Challenges in Information Science - 19th International Conference, RCIS 2025, Seville, Spain, May 20-23, 2025, Proceedings, Part I (LNCS, Vol. 547)*. Springer, 245–261.

[8] Pavel Koupil, Jáchym Bártík, and Irena Holubová. 2022. MM-evocat: A Tool for Modelling and Evolution Management of Multi-Model Data. In *Proc. of CIKM '22*. ACM, 4892–4896.

[9] Pavel Koupil, Daniel Crha, and Irena Holubová. 2025. A universal Approach for Simplified Redundancy-Aware Cross-Model Querying. *Information Systems* 127 (2025).

[10] Pavel Koupil and Irena Holubová. 2022. A Unified Representation and Transformation of Multi-Model Data using Category Theory. *J. Big Data* 9, 1 (2022), 61.

[11] Pavel Koupil, Sebastián Hricko, and Irena Holubová. 2022. MM-infer: A Tool for Inference of Multi-Model Schemas. In *Proc. of EDBT '22*. OpenProceedings.org, 2:566–2:569.

[12] Pavel Koupil, Martin Svoboda, and Irena Holubová. 2021. MM-cat: A Tool for Modeling and Transformation of Multi-Model Data using Category Theory. In *Proc. of MODELS '21*. IEEE, New York, NY, USA, 635–639.

[13] Jiaheng Lu and Irena Holubová. 2019. Multi-model Databases: A New Journey to Handle the Variety of Data. *ACM Comput. Surv.* 52, 3, Article 55 (2019), 38 pages.

[14] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-Sructured Deep Neural Network Models for Query Performance Prediction. *Proc. VLDB Endow.* 12, 11 (July 2019), 1733–1746.

[15] Uta Störl and Meike Klettke. 2022. Darwin: A Data Platform for Schema Evolution Management and Data Migration. In *Proceedings of the Workshops of the EDBT/ICDT 2022 Joint Conference, Edinburgh, UK, March 29, 2022 (CEUR Workshop Proceedings, Vol. 3135)*. CEUR-WS.org.

[16] Uta Störl, Meike Klettke, and Stefanie Scherzinger. 2020. NoSQL Schema Evolution and Data Migration: State-of-the-Art and Opportunities (Tutorial). In *Proc. of EDBT '20*. OpenProceedings.org, 655–658.

[17] Jun Sun, Feng Ye, Nadia Nedjah, Ming Zhang, Dong Xu, and Yu-an Tan. 2023. Workload-Aware Performance Tuning for Multimodel Databases Based on Deep Reinforcement Learning. *Int. J. Intell. Syst.* 2023 (Jan. 2023), 17 pages.

[18] Feng Ye, Yang Li, Xiwen Wang, Nadia Nedjah, Peng Zhang, and Hong Shi. 2022. Parameters tuning of multi-model database based on deep reinforcement learning. *J. Intell. Inf. Syst.* 61, 1 (Nov. 2022), 167–190.