# *TransforMMer*: A Universal Multi-Model Data Generator

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

Alžběta Šrůtková
Charles University
Prague, Czech Republic
srutkova.alzbeta@gmail.com

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

## ABSTRACT

The **V**ariety aspect of Big Data poses challenges for most established single-model data management solutions. However, as multi-model systems increase, especially in the data-management domain, the demand for benchmarking and comparison grows. However, real-world multi-model datasets are scarce, and existing benchmarks are limited in number and versatility. This demo paper introduces *TransforMMer*, a tool to generate pseudo-realistic multi-model data from given single-model (or multi-model) datasets. It is based on a pipeline leveraging a subset of our multi-model data management tools to automate schema inference, enable user-defined schema modifications, and generate data reflecting these changes. In the demonstration, we will introduce the tool, the repository of created multi-model datasets, their challenges, and future extensions.

## 1 INTRODUCTION

The traditional relational data model has long been the standard for data representation, but Big Data has revealed its limitations. A key challenge is the **V**ariety of Big Data, which encompasses multiple types and formats that originate from diverse sources and are inherently adherent to different models. There are structured, semi-structured, and unstructured formats; order-preserving and order-ignorant models; aggregate-ignorant and aggregate-oriented systems; etc. The naturally contradictory features of the so-called *multi-model data*, combining distinct models with contradictory features, complicates all aspects of data management, from modelling to querying.

Many tools now support multi-model capabilities – e.g., more than two-thirds of the top 50 database management systems (DBMSs)[1] align with Gartner's decade-old prediction of support for multiple models [2]. However, the lack of standards for combining models leaves each DBMS with proprietary solutions. Even the variety of query languages [4] highlights the state of multi-model data systems.

Selecting the optimal tool for a specific use case is challenging, and extensive benchmarking is essential to compare tools across all target use cases. While single-model benchmarks and data generators exist for standard models, their adaptation to the multi-model paradigm is complex. Multi-model test cases must account for numerous combinations of different models using various strategies such as embedding, cross-model references, or redundancy. And this limits the availability and versatility of truly comprehensive multi-model benchmarks. [10, 12]

Let us illustrate the problem with an example. We are building a web application that depends on various types of data. When it comes to selecting a database solution, we face a dilemma. We want the reliability of a relational database (e.g., for products and orders), but we also need to be able to perform complex graph queries (e.g., about the users, their mutual connections, and interactions with our products). We can leverage a multi-model database to store such data; however, we are uncertain whether this approach is more efficient than storing everything in a single database system. Many single-model datasets are similar to our expected data, so we can use them to benchmark the performance of the single-database solution. However, there are very few multi-model datasets available, nor there are tools for generating them.

To address this challenge, we introduce *TransforMMer*, a tool for generating virtually any multi-model dataset. Instead of relying on traditional value generators with predefined distributions, our approach uses real-world single-model (or multi-model) datasets as input. It infers a schema of the input data, enables its user-specified modification to reflect the required combination of models, and transforms the output to the respective form. It also supports redundancy and versioning, which further increase its versatility. For its implementation, we integrate and extend selected tools from our toolset based on the unifying categorical representation of multi-model data [7]. Combining and extending tools for modelling and transformations [9], schema inference [8], and evolution management [6] with appropriate integration modules and interfaces allows us to ensure the transformation universally and correctly.

*Outline.* In Section 2, we review related work. Section 3 introduces the categorical representation of multi-model data. Section 4 introduces *TransforMMer* and Section 5 its demonstration.

## 2 RELATED WORK

Two main approaches to benchmarking data management tools involve real-world data sets or synthetic data generators. Many focus on single models, but there are very few multi-model options.

Real-world datasets are primarily relational due to the dominance of relational DBMSs, accompanied by hierarchical models (e.g., JSON for NoSQL DBMSs) and graph data. Popular sources include Kaggle[2] or Harvard Dataverse.[3] Graph data can be found in collections like SNAP.[4] Governments (e.g., US,[5] EU,[6] etc.) provide open data portals involving distinct data formats. Moreover, various datasets can also be found on GitHub or Google Dataset Search.

However, often we cannot easily find a suitable real-world dataset, and we need to use a data generator or benchmark. However, most are limited to specific data models, formats, or fixed use cases. For example, TPC-H and TPC-DS[7] focus on the relational model. Benchmarks such as XMark [11] are tailored to the document model. And also, many graph data generators exist [1].

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

[2] https://www.kaggle.com/
[3] https://dataverse.harvard.edu/
[4] https://snap.stanford.edu/data/
[5] https://data.gov/
[6] https://data.europa.eu/
[7] https://www.tpc.org/

Regarding multi-model data, BigBench [3] covers semi-structured and unstructured data, but lacks support for graph and array data models. UniBench [12] does not support the array data model either, and it considers only a single use case. M2Bench [5] encompasses relational, document, graph, and array data models but also involves only three predefined use cases.

## 3 CATEGORICAL MODEL OF DATA

First, to unify the terminology from different models, we use the following terms: A *kind* corresponds to a class of items (e.g., a relational table or a collection of JSON documents). A *record* corresponds to one item of a kind (e.g., a table row or a JSON document). A record consists of simple or complex *properties* having their *domains*.

We also recall the basic notions of category theory. A *category* $\mathbf{C} = (O, \mathcal{M}, \circ)$ consists of a set of objects $O$, set of morphisms $\mathcal{M}$, and a composition operation $\circ$ over the morphisms ensuring transitivity and associativity. Each morphism is modelled as an arrow $f : A \to B$, where $A, B \in O$, $A = dom(f), B = cod(f)$. And there is an *identity* morphism $1_A \in \mathcal{M}$ for each object $A$. A category can be visualized as a multigraph, with objects represented as vertices and morphisms as directed edges.

The core and the integration point of all our tools is an abstract representation of multi-model data called *schema category* [7]. It is defined as a tuple $\mathbf{S} = (O_\mathbf{S}, \mathcal{M}_\mathbf{S}, \circ_\mathbf{S})$. *Objects* in $O_\mathbf{S}$ correspond to the domains of the ER model's entity types, attributes, and relationship types. Each schema object $o \in O_\mathbf{S}$ is internally represented as a tuple $(key, label, superid, ids)$, where $key$ is an automatically assigned internal identity, $label$ is an optional user-defined name, $superid \neq \emptyset$ is a set of attributes (each corresponding to a signature of a morphism) forming the actual data contents a given object is expected to have, and $ids \subseteq \mathcal{P}(superid), ids \neq \emptyset$ is a set of particular identifiers (each modeled as a set of attributes) allowing us to distinguish individual data instances uniquely. *Morphisms* connect appropriate pairs of objects. Each morphism $m \in \mathcal{M}_\mathbf{S}$ is represented as a tuple $(signature, dom, cod, label)$. The *signature* allows us to distinguish all morphisms except the identity ones mutually. *dom* and *cod* represent the domain and codomain of the morphism. Finally, $label \in \{$ `#property`, `#role`, `#isa`, `#ident` $\}$ allows us to further distinguish morphisms with semantics "has a property", "has an identifier", "has a role", or "is a". The explicitly defined morphisms are denoted as *base*, obtained via the composition $\circ$ as *composite*.

The decomposition of a schema category $\mathbf{S}$, eventually partial or overlapping, is defined via a set of *mappings*. For each kind, the mapping specifies where and how its records are stored in a selected single-/multi-model DBMS using a so-called *access path*, which recursively describes the structure of a kind.

For example, in Fig. 3, we can see a sample schema category with kinds *User*, *Business* and *Tip* having root objects denoted with green. They are mapped to JSON document collections from Fig. 1. For simplicity, we omit the most common edge label "has a property" and replace uninteresting properties with dots in an oval.

## 4 TRANSFORMMER

To solve the indicated open problems, *TransforMMer* enables the transformation of virtually any input dataset (single- or multi-model) to a required output multi-model dataset.

### 4.1 Sample Scenario

Let us consider a simple scenario to demonstrate the purpose and advantages of *TransforMMer*. We want to use the subset of the Yelp Open Dataset,[8] namely the data that describes businesses, tips, and users, represented using the JSON format as depicted in Fig. 1.

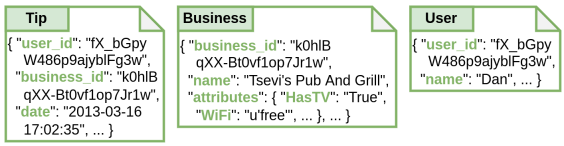


**Figure 1: Input Yelp dataset in the document model**

We want to transform the data into a combination of the relational and graph models, e.g., the one depicted in Fig. 2.

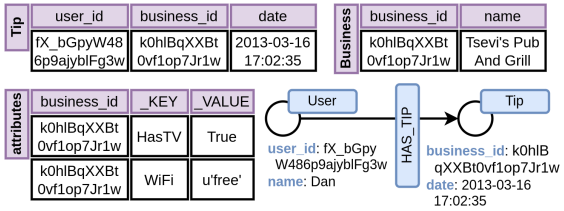


**Figure 2: Yelp dataset in the graph and relational model**

*4.1.1 Initial Schema Category.* *TransforMMer* first infers the *initial schema category* $\mathbf{S_{ini}}$ depicted for the input data in Fig. 3. As we can see, each original JSON document corresponds to one kind in the schema category, namely *User*, *Business*, and *Tip*.

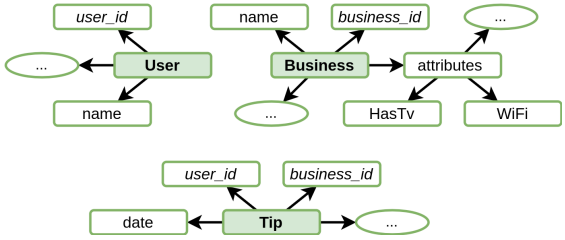


**Figure 3: Initial schema category $\mathbf{S_{ini}}$ inferred from the sample data in Fig. 1**

*4.1.2 Improved Schema Category.* As $\mathbf{S_{ini}}$ is very simple and even incorrect, *TransforMMer* enables its improvements. During the process of inference of $\mathbf{S_{ini}}$, it also infers a list of candidates for basic integrity constraints (identifiers and references), clustering, or recursion. The user can confirm or refuse them or add his/her suggestions to create an improved schema category $\mathbf{S_{imp}}$.

*Identifiers.* First, we need an identifier for each kind to create a correct schema category. The inference process of *TransforMMer* creates a list of simple properties that are candidate identifiers, i.e., they have a unique and compulsory value in each respective record. The user can select the correct ones from them. The candidates do not include composite identifiers, as finding all possible combinations would be too computationally expensive. However, the user can define composite identifiers manually.

[8] https://www.yelp.com/dataset

In the sample $S_{ini}$ in Fig. 3, kind *Business* has only one candidate identifier *business_id*, so we choose it as its identifier. Kind *User* also has only one candidate identifier *user_id*. Kind *Tip* does not have any (simple) candidate identifier, so we create a composite identifier consisting of properties *user_id* and *business_id*. (The identifiers are depicted in Fig. 4 in curly brackets in root nodes of the kinds.)

*References.* Next, we can define references between the kinds to the respective identifiers. *TransforMMer* also automatically infers candidates for the references (based on the inclusion of values of the referenced properties).

In the sample $S_{ini}$ in Fig. 3, we identify a reference from kind *Tip* to kinds *User* and *Business* based on the values of properties *user_id* and *business_id*. The result is depicted in Fig. 4, where the two objects *user_id* were merged into one. The same happened for objects *business_id*. However, the primary identifier of kind *Tip* is still the same; it is just represented by different schema morphisms.

*Clustering of Properties.* Further improvements in the schema category involve the integration of various more complex constructs. One of them is replacing (huge) clusters of properties with maps. According to our analyses of real-world data, they may often contain multiple properties which have the same structure but different names (such as the property *attributes* of kind *Business* depicted in Fig. 3). They make the schema unnecessarily complex, so *TransforMMer* allows the user to transform them into a single set of key-value pairs. The user must just select all the properties to be represented as a map.

The result is depicted in Fig. 4. As we can see, the direction of morphism between objects *Business* and *attributes* changed. This is because initially, the object *attributes* corresponded to a complex property containing all the attributes, but now, it represents an array of key-value pairs. So, the cardinality specified by the morphism changed, and the morphism had to be updated (for details, see [7]).

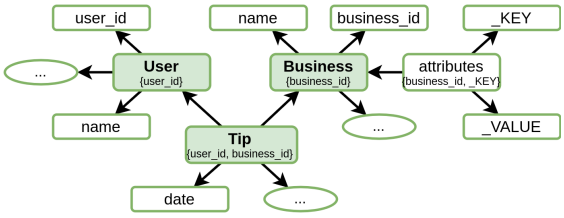


**Figure 4: The improved schema category $S_{imp}$ of the Yelp data containing identifiers, a reference, and a map of properties**

*Recursion.* Similarly to clustering of properties, i.e., generally discovering a repeating pattern, we can also identify the recursion, i.e., a pattern occurring not within the same parent property but mutually nested. For example, consider a complex property *comment* consisting of properties *author*, *content*, and *replies*, where property *replies* is an array of properties *comment*. Such structure can be repeated in the inferred schema as many times as is the maximum depth of recursion in the source data.

We can significantly simplify this schema by defining a pattern for the property *comment*. *TransforMMer* then creates only a single complex property for each unique property in the pattern and a morphism from property *replies* to property *comment*. Thus, the resulting schema category contains a cycle, allowing infinite recursion. An example of such a transformation is depicted in Fig. 5, which simultaneously provides sample screenshots of *TransforMMer*.

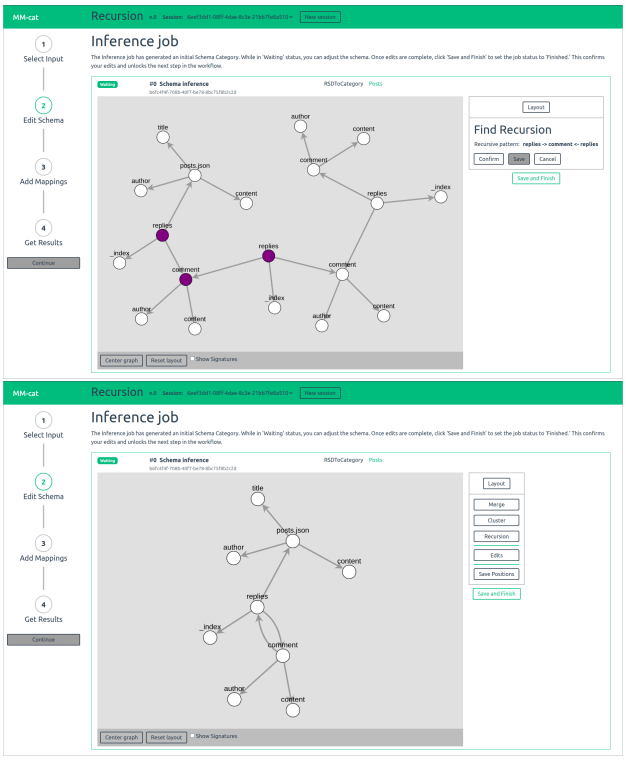


**Figure 5: Screenshot of *TransforMMer* before and after the elimination of a recursive structure**

*4.1.3 Mapping Modification.* Having the final improved schema category $S_{imp}$ (see Fig. 4), the user can now define its new mapping. As depicted in Fig. 6, for example, the original document model (green) can be entirely discarded, and instead of it, the user may define four new kinds. Kind *User* is mapped to the graph model (blue), where each user represents a node in the graph, and edges represent mutual friendship. Kinds *Business*, *Tip*, and *attributes* are mapped to the relational model (purple), i.e., tables. Considering the relational model without arrays, the kind *attributes* had to be established. The output data are depicted in Fig. 2.

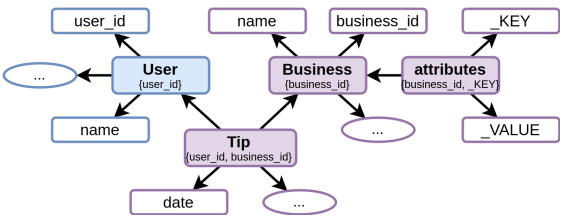


**Figure 6: $S_{imp}$ of the Yelp data re-mapped to the graph (blue) and relational (purple) model to represent data in Fig. 2**

## 4.2 Workflow

The whole process of transforming the input (single- or multi-model) data to required multi-model data is depicted in Fig. 7.

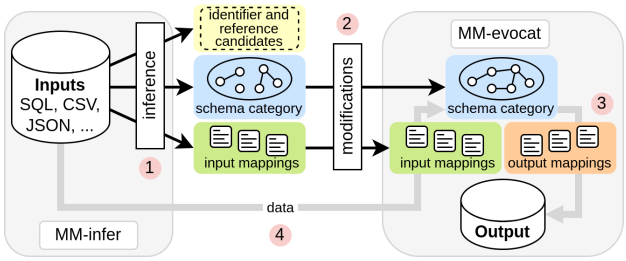


**Figure 7: The workflow of *TransforMMer***

*Step 1.* First, the users need to specify input data sources. *TransforMMer* supports the input from a file system, a DBMS, or their combination. It then infers the schema of each kind (table, collection, etc.) in the data sources. The output of this step is $\mathbf{S_{ini}}$ and its mapping to the logical structures of the original data. The algorithm also infers candidates for identifiers, references between the kinds, and repeating patterns.

*Step 2.* $\mathbf{S_{ini}}$ is improved by interacting with the user, i.e. $\mathbf{S_{imp}}$ is created. The user can specify identifiers, references, and repeating patterns to be merged from the candidate set or his/her own. The user can also further modify the schema category, e.g., by grouping a set of properties into a new property or deleting a property. With every change, *TransforMMer* automatically updates the mappings.

*Step 3.* $\mathbf{S_{imp}}$ is mapped to the new requested combination of models. The user selects parts of the schema category and specifies the requested, possibly overlapping mapping (e.g., by modifying an existing one or creating a new one from scratch). *TransforMMer* ensures all the respective modifications.

*Step 4. TransforMMer* loads the original data according to the input mappings into an in-memory categorical representation and then stores it according to the new user-specified mappings into the output. It can be stored in a specified file system or a DBMS.

### 4.3 Architecture

As mentioned in the introduction, *TransforMMer* leverages selected members of our toolset for multi-model data management, namely *MM-infer* [8] for categorical schema inference, *MM-evocat* [6] for schema modifications, and *MM-cat* [9] (now a part of *MM-evocat*) for schema modelling and data transformations. Fig. 7 also indicates their integration in *TransforMMer*. *TransforMMer* consists of the backend (written in Java), which contains all the algorithms, and a single-page client (written in Vue), which provides the user interface. The code repository is available at GitHub.[9] And a demo is also available.[10] Lastly, we have a short video tutorial.[11]

All of the algorithms mentioned above use abstract wrappers, which provide a unified interface to the data sources instead of working with them directly. This allows us to easily add support for new data sources by implementing a new wrapper. For now, we support PostgreSQL, MongoDB, Neo4j (i.e., representatives of the relational, document, and graph model), CSV and JSON files (i.e., the most common file formats).

[9] https://github.com/mmcatdb/mmcat
[10] https://demo.mmcatdb.com
[11] https://www.youtube.com/watch?v=Aiw815OaAb4

Because the users might want to work with large datasets, all processes are implemented as asynchronous jobs.

### 4.4 Dataset Repository

As a proof of concept, we have used *TransforMMer* to generate a repository[12] of popular multi-model datasets (to be gradually extended). The current datasets involve, e.g., the Yelp dataset, the IMDb dataset,[13] or the dataset of NASA open-source code projects.[14] In the next steps, we want to extend the repository with the popular datasets and their variations and using a comparative survey demonstrate the wide usability of *TransforMMer* for comparing multi-model databases.

## 5 DEMONSTRATION OUTLINE

In our presentation, we will first demonstrate the described functionality of *TransforMMer* using the Yelp dataset. We will also introduce the repository of multi-model data sets created using *TransforMMer* and show selected specifics and challenges we encountered in the real-world datasets, the support for redundancy, and the applicability of versioning to simulate data evolution. Finally, we will overview future plans for the generation of respective queries. The interested audience can also experiment with *TransforMMer* and own datasets.

## ACKNOWLEDGEMENT

Supported by the GAČR grant no. 23-07781S, GAUK grant no. 292323., and SVV project no. 260 698.

## REFERENCES

[1] Angela Bonifati, Irena Holubová, Arnau Prat-Pérez, and Sherif Sakr. 2020. Graph Generators: State of the Art and Open Challenges. *ACM Comput. Surv.* 53, 2, Article 36 (apr 2020), 30 pages.
[2] Donald Feinberg, Merv Adrian, Nick Heudecker, Adam M. Ronthal, and Terilyn Palanca. 12 October 2015. Gartner Magic Quadrant for Operational Database Management Systems, 12 October 2015.
[3] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. BigBench: towards an industry standard benchmark for big data analytics. In *SIGMOD '13*. ACM, 1197–1208.
[4] Qingsong Guo, Chao Zhang, Shuxun Zhang, and Jiaheng Lu. 2023. Multi-model query languages: taming the variety of big data. *Distributed and Parallel Databases* (31 May 2023).
[5] Bogyeong Kim, Kyoseung Koo, Undraa Enkhbat, Sohyun Kim, Juhun Kim, and Bongki Moon. 2022. M2Bench: A Database Benchmark for Multi-Model Analytic Workloads. *Proc. VLDB Endow.* 16, 4 (dec 2022), 747–759.
[6] Pavel Koupil, Jáchym Bártík, and Irena Holubová. 2022. MM-evocat: A Tool for Modelling and Evolution Management of Multi-Model Data. In *CIKM '22*. ACM, 4892–4896.
[7] 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.
[8] Pavel Koupil, Sebastián Hricko, and Irena Holubová. 2022. MM-infer: A Tool for Inference of Multi-Model Schemas. In *EDBT '22*. OpenProceedings.org, 2:566–2:569.
[9] Pavel Koupil, Martin Svoboda, and Irena Holubová. 2021. MM-cat: A Tool for Modeling and Transformation of Multi-Model Data using Category Theory. In *MODELS '21*. IEEE, New York, NY, USA, 635–639.
[10] 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 (June 2019), 38 pages.
[11] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. 2002. XMark: a benchmark for XML data management. In *VLDB '02*. VLDB Endowment, 974–985.
[12] Chao Zhang, Jiaheng Lu, Pengfei Xu, and Yuxing Chen. 2019. UniBench: A Benchmark for Multi-model Database Management Systems. In *TPCTC 2018*. Springer International Publishing, Cham, 7–23.

[12] https://dare.mmcatdb.com
[13] https://developer.imdb.com/non-commercial-datasets/
[14] https://data.nasa.gov/Software/NASA-open-source-code-projects-with-A-I-generated-/3efg-u4v8/about-data