# Unlocking Optimal ORM Database Designs: Accelerated Tradeoff Analysis with Transformers

MD RASHEDUL HASAN, University of Nebraska-Lincoln, USA
MOHAMMAD RASHEDUL HASAN, University of Nebraska-Lincoln, USA
HAMID BAGHERI, University of Nebraska-Lincoln, USA

Optimizing object-relational database mapping (ORM) design is crucial for performance and scalability in modern software systems. However, widely used ORM tools offer limited support for exploring performance tradeoffs, often enforcing a single design and overlooking alternatives, which can lead to suboptimal outcomes. While systematic tradeoff analysis can reveal Pareto-optimal designs, its high computational cost and poor scalability hinder practical adoption. This paper presents DesignTradeoffSculptor, an extensible tool suite for efficient, scalable tradeoff analysis in ORM database design. Leveraging advanced Transformer-based deep learning models—trained and fine-tuned on formally analyzed database designs—and framing design exploration as a Natural Language Processing task, DesignTradeoffSculptor efficiently identifies and removes suboptimal designs, sharply reducing the number of candidates requiring costly tradeoff analysis. Experiments show that DesignTradeoffSculptor uncovers optimal designs missed by leading ORM tools and improves analysis efficiency by over 98.21%, reducing tradeoff analysis time from 15 days to just 18 minutes, demonstrating the transformative potential of integrating formal methods with deep learning.

CCS Concepts: • **Software and its engineering → Software design tradeoffs**.

Additional Key Words and Phrases: software design, tradeoff analysis, machine learning

**ACM Reference Format:**
Md Rashedul Hasan, Mohammad Rashedul Hasan, and Hamid Bagheri. 2025. Unlocking Optimal ORM Database Designs: Accelerated Tradeoff Analysis with Transformers. 2, FSE, Article FSE074 (July 2025), 24 pages. https://doi.org/10.1145/3729344

## 1 INTRODUCTION

Contemporary software systems are intrinsically reliant on the management and retrieval of extensive data. Consequently, a pivotal design decision in developing such systems is how to structure the data storage and its interaction with the software. As of August 2024, relational databases continue to maintain their dominance in the database market [15, 49]. These databases rely on a relational database management system (RDBMS), such as Oracle or MySQL, to store and retrieve data. Designing a highly performant and scalable database poses a significant challenge, even for well versed database architects. It becomes an even more daunting task when undertaken by software engineers who are primarily trained in, and experienced with, object-oriented software development paradigms. This challenge, often referred to as the *object-relational impedance mismatch*

Authors' addresses: Md Rashedul Hasan, University of Nebraska-Lincoln, Lincoln, USA, mhasan6@huskers.unl.edu; Mohammad Rashedul Hasan, University of Nebraska-Lincoln, Lincoln, USA, hasan@unl.edu; Hamid Bagheri, University of Nebraska-Lincoln, Lincoln, USA, bagheri@unl.edu.


XXXX-XXXX/2025/7-ARTFSE074
https://doi.org/10.1145/3729344

*problem* [13, 26, 27], has persisted since the emergence of object-oriented programming techniques and continues to be a striking concern today [13].

Several object-relational mapping (ORM) middleware tools, such as SQLAlchemy [48], Hibernate [18], Entity Framework [38], and Django ORM [17], aim to assist developers to bridge this mismatch gap. These tools automate the generation of relational tables and queries tailored to the specific RDBMS used by a software system, based on annotated software objects or configuration files. However, they often provide limited insights into the performance implications of different mapping strategies. Instead, they tend to lock developers into using a single mapping strategy, without considering the available object-relational mapping options or the tradeoffs they entail.

Prior studies have demonstrated that systematic tradeoff analysis is a valuable approach for guiding the design of object-relational database mappings, not only offering insights into optimal design choices but also unveiling more efficient and optimized design alternatives that are often overlooked by widely-used leading ORM design techniques [6, 7]. However, it comes with inherent limitations. The systematic tradeoff analysis, while informative, tends to be slow and resource-intensive. The substantial expense associated with identifying and synthesizing the space of alternatives, despite significant research efforts [51–53, 61, 62, 66], as well as assessing them, limits its practical scalability. Additionally, the computational overhead associated with evaluating individual design alternatives can significantly impede the efficiency of systematic tradeoff analysis. For example, applying a state-of-the-art tradeoff analysis technique required over 15 days on average solely to evaluate the database design of a system under examination (cf. Section 4). These challenges underscore the need for efficient and scalable approaches to tradeoff analysis in the context of object-relational database design.

To address this situation, in this paper, we introduce DesignTradeoffSculptor, a novel approach and accompanying tool suite designed to enable efficient and scalable tradeoff analysis for identifying optimal database designs in object-oriented software systems. DesignTradeoffSculptor relieves developers from the burden of conducting prohibitively expensive evaluations for every single point within the entire design space, thereby facilitating the practical application of systematic tradeoff analysis in real-world software system development. Our approach leverages cutting-edge deep learning (DL) methods to pin down the database designs that are most likely to yield (Pareto-)optimal[1] tradeoffs for developers, thereby excluding inferior design points from subsequent analysis. The crux of our approach revolves around the training of a DL model using a comprehensive dataset comprising formally analyzed database designs. This meticulously trained model is subsequently deployed to identify (Pareto-)optimal tradeoffs within the design space, thereby substantially curtailing the number of candidate designs necessitating dynamic evaluation. We employ DL techniques to efficiently navigate the intricacies of the design space. Our approach involves framing design space analysis as a Natural Language Processing (NLP) problem, utilizing state-of-the-art Transformer-based DL models [60] to ascertain the Pareto-optimality of designs across the entire space. Through rigorous experimentation, we explore a variety of Transformer models and learning strategies, aiming to pinpoint the most effective approach for comprehensive design space analysis. These curated design selections can then proceed to the evaluation and comparative analysis phase, culminating in the presentation of the top-performing designs to the user. This streamlined approach significantly accelerates the tradeoff analysis process in comparison to the state-of-the-art techniques, saving remarkable time and resources.

[1]Pareto-optimal, a term derived from the Pareto principle, refers to solutions that cannot be improved in one aspect without sacrificing another, thereby representing the highest degree of efficiency attainable within a specific context.

We thoroughly evaluate DesignTradeoffSculptor across a diverse range of software database systems originating from various domains. The experimental results strongly support the effectiveness of tradeoff analysis in revealing Pareto-optimal design schemas that have previously gone unnoticed by the leading ORM design tools [17, 18, 38, 48]. This underscores the competence of DesignTradeoffSculptor to unveil more efficient and optimized design alternatives, which are often overlooked by widely-used state-of-the-practice techniques, including SQLAlchemy [48], Hibernate [18], Entity Framework [38], and Django ORM [17]. Furthermore, our results reveal a remarkable enhancement in analysis efficiency, with an average improvement of over 98.21% compared to the current state-of-the-art in tradeoff analysis. Notably, this translates to a reduction in the time required for tradeoff analysis from the arduous 15 days, as required by the leading method in tradespace analysis, to a mere 18 minutes, marking a substantial leap in efficiency (cf. Section 4). These findings corroborate the significant contributions of DesignTradeoffSculptor in revolutionizing the landscape of database design and analysis.
In summary, this paper makes the following contributions.

- *Efficient Design Tradespace Analysis, Bridging Formal Methods and Machine Learning:* This paper introduces a novel approach that combines the validity of formal methods with the scalable convenience of machine learning, resulting in a pragmatic solution for efficient database design.
- *DesignTradeoffSculptor Implementation:* We develop DesignTradeoffSculptor as an extensible tool, backed by training a transformer model on a dataset of formally analyzed database designs to overcome the limitations of resource-intensive design tradeoff analysis methods, making them practical for real-world software development in database design. We make research artifacts and experimental data available to the research and education community [1].
- *Experimental Validation:* We conduct a thorough evaluation of DesignTradeoffSculptor across various software database systems from diverse domains. The results demonstrate its effectiveness in revealing Pareto-optimal design schemas that often go unnoticed by leading database design tools. We achieve an average improvement of over 98.21% in analysis efficiency compared to the current state-of-the-art, revolutionizing the landscape of database design and analysis.

## 2 MOTIVATING EXAMPLE

To motivate the research and illustrate our approach, we provide an example of the driving problem tackled in this paper: assisting software designers in developing the optimal database designs for their object-oriented software systems. We begin by briefly introducing the widely adopted state-of-the-practice tools, i.e., object-relational mapping (ORM) tools such as Hibernate [18]. These tools typically provide limited, single-point ORM designs to developers, offering minimal guidance in the selection of the optimal design. We then delve into the state-of-the-art techniques aimed at aiding designers in analyzing the tradeoffs within database design. While effective in presenting choices to developers, these methods encounter significant challenges when applied to large, real-world software systems. Our approach, which we elaborate on in Section 3, introduces an innovative solution to this scalability issue. Leveraging machine learning, our approach identifies superior candidate solutions without the need for evaluating the entire tradeoff space. This not only considerably expedites the analysis process compared to the state-of-the-art but also ensures the quality of the results.

**Database Design Challenges in Practice.** The development of software systems often involves working with data, and one common practice is to use an RDBMS for data storage and retrieval. While there are many well-established RDBMS solutions available, they are inherently relational, which can pose challenges for software developers who are more familiar with object-oriented (OO)

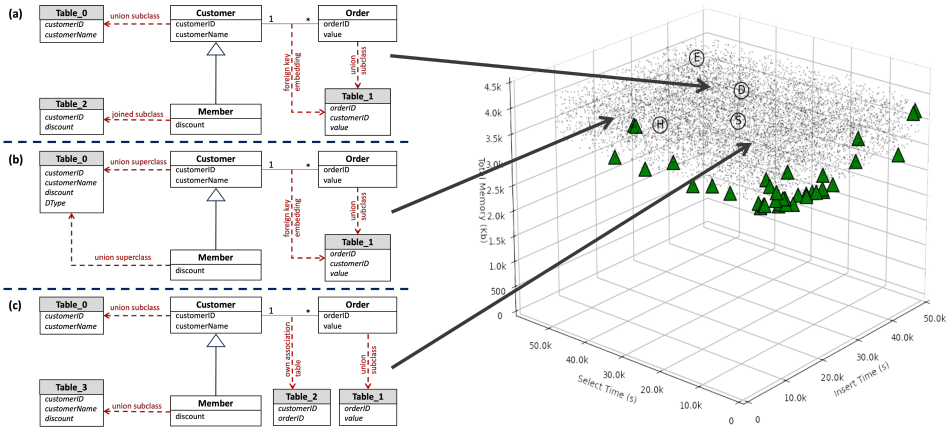


Fig. 1. Partial design space with quality attributes (i.e., tradespace) for database designs for an E-commerce system, comparing insert and select time and required storage space. (a)-(c) depict partial object-relational mappings for three design alternatives, with arrows in the scatter plot indicating their positions (black dots represent individual design alternatives). Designs with optimal tradeoffs (Pareto optimal) are indicated by triangles. Design solutions generated by state-of-the-practice object-relational mapping tools, namely SQLAlchemy [48], Hibernate [18], Entity Framework [38], and Django ORM [17], are represented respectively by Ⓢ, Ⓗ, Ⓔ, and Ⓓ. Note that the designs produced by widely-used state-of-the-practice tools are noticeably is far from optimal, and only a small fraction of designs present optimal tradeoffs. Systematic synthesis and evalution of the entire model space is required to find the optimal tradeoffs.

paradigms. This disconnect has led to the emergence of the object-relational impedance mismatch problem, necessitating developers to be proficient in both OO and relational concepts. To address this issue, various object-relational mapping tools, like Hibernate and Entity Framework, have been introduced. These ORM tools enable developers to map object-oriented data models (OODM) onto relational tables using different mapping strategies. For instance, Figures 1(a)-(c) present three distinct mappings for a segment of an ecommerce OODM, which includes Customer, Member, and Order. These mappings employ varying strategies to represent the association between Customer and Order and the inheritance relationship between Customer and Member. The data model is represented by white boxes, with table headers highlighted in grey. Each ORM employs mapping strategies, represented by dashed lines, to depict the association between Customer and Order, as well as the inheritance relationship between Customer and Member, within the relational tables.

However, despite the widespread use of ORM tools, the impedance mismatch problem persists, particularly in selecting the most suitable mapping strategies to meet a system's non-functional requirements, which remains a challenging task with limited assistance from the current state of practice. While ORM tools provide relief from many runtime-related tasks, they still require developers to define application-specific data mapping specifications. This task involves selecting mapping strategies to transform an OODM into a relational database schema, each with varying impacts on non-functional properties like performance and scalability. Many existing ORM solutions adopt a one-size-fits-all approach, choosing a single mapping strategy without considering the array of available options and the associated tradeoffs. This leaves developers with the intricate task of choosing the best mapping strategies based on their application's specific non-functional requirements, necessitating a deep understanding of both object-oriented and relational paradigms, as well as the tradeoffs involved in making these choices within the vast design space.

**Challenges in State-of-the-Art Tradeoff Analysis.** To address the limitations of state-of-the-practice ORM techniques, software researchers employ systematic tradeoff approaches to

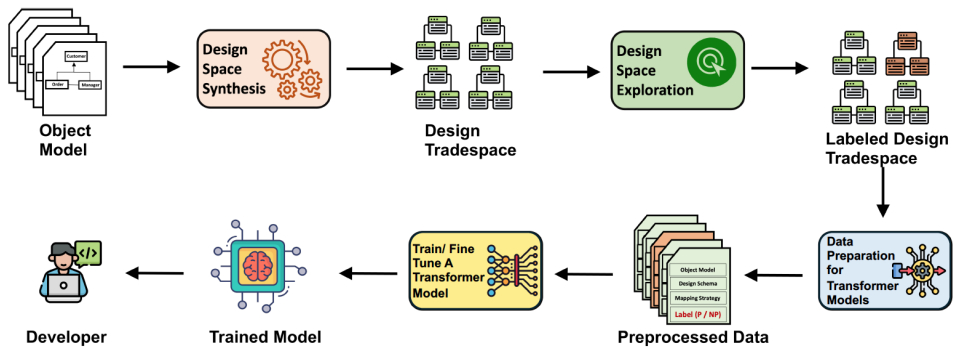


Fig. 2. Approach Overview.

evaluate and compare various database designs, seeking optimal performance tailored to their specific needs. A prominent state-of-the-art technique for assessing the quality of database designs in object-oriented software systems is specification-driven tradespace synthesis [6, 7]. It takes an OODM specification of the system to be analyzed, denoted as $\mathcal{S}$, as its input. The system specification $\mathcal{S}$, along with the formally specified constraints of the generic mapping strategies, denoted as $\mathcal{I}$, collectively sets the context and constraints for the tradespace exploration. These specifications are passed into an off-the-shelf solver/model finder (e.g., Alloy Analyzer [28]); the solver seeks a system-specific ORM model, denoted as $m$, that satisfies $\mathcal{S} \wedge \mathcal{I}$. Each such model is used to generate a corresponding design variant, enriching the tradespace. Whenever a model $m$ is found, the system specification $\mathcal{S}$ is updated to exclude $m$, and the solver is rerun on this updated specification to find additional valid models, and consequently, more design alternatives. When no more satisfying models can be found, the synthesis process concludes, having exhaustively generated designs that align with the input specification. Subsequently, each generated design is dynamically assessed to determine the values of associated design *attributes*, positioning it within the tradespace. The synthesized designs are subsequently compared with each other to identify the *Pareto frontier* within the tradespace. This frontier represents points where no single attribute can be improved without compromising others, signifying a well-balanced tradeoff among design attributes. Figure 1 illustrates an example tradespace for an E-commerce system. However, this process is time-intensive. For instance, when applied to the E-commerce system OODM, it demands almost 66 days of computation time in total.

This example points to one of the most challenging issues in software design tradeoff analysis, i.e., while the state-of-the-art in the tradespace analysis can reliably produce extensive design tradeoff spaces and their Pareto frontiers, the process is prohibitively costly and time-consuming. In the next sections, we first provide an overview of DesignTradeoffSculptor and then delve into more details about its approach to address this issue.

## 3 APPROACH

This section provides an in-depth presentation of DesignTradeoffSculptor, which automates the identification of optimal database designs in object-oriented software systems. Figure 2 depicts a high-level overview of the approach, which consists of two primary phases: training and prediction. *(1) Training Phase:* The approach commences with the formal and precise synthesis and exhaustive evaluation of all database designs for a given set of training object data models, utilizing state-of-the-art tradespace analysis techniques. These rigorously constructed design tradespaces, including associated values for relevant design attributes, serve as training data for a transformer model.

*(2) Prediction Phase:* When presented with a novel object data model, DesignTradeoffSculptor synthesizes its design space but bypasses the exhaustive and costly evaluation. Instead, the design space is fed into the previously trained Transformer-based DL model. This model efficiently identifies potential Pareto-optimal candidates, which are then dynamically evaluated. This process significantly reduces the number of design candidates requiring further assessment, allowing for the determination of final Pareto-optimal designs without the need for expensive exhaustive analysis. The approach culminates in presenting these Pareto-optimal designs to the end-user, offering efficient and optimized database design solutions for new object models.

### 3.1 Design Tradespace Synthesis

The initial phase of DesignTradeoffSculptor involves collecting database design tradeoff spaces as training data. To clarify our methodology, we define several key terms. The object model specification is a formal description of the subject system's structure, detailing classes, attributes, relationships, and constraints. Design attributes are quantifiable performance metrics used to evaluate each database design and include query selection time (the duration required to execute queries), memory usage (the storage space consumed by the database), and insertion time (the time taken to add new records). A database design tradeoff space refers to the set of possible database designs for a given object model, along with their associated performance metrics.

To ensure the quality of our investigation, we took inspiration from the state-of-the-art solution called Trademaker [7], known for its ability to formally synthesize exhaustive database designs for a given object model. Using a similar procedure, DesignTradeoffSculptor generates an extensive set of database designs, which form the basis for our analysis. Specifically, it translates the provided object model specification into Alloy's relational formula [29], and then employs an off-the-shelf SAT solver to produce satisfying design models. Once a database design model is discovered, it is added to the design space. It then negates each discovered model and incorporates these negations back into the relational formula. It subsequently iterates through this process of design model synthesis, aiming to identify models for the updated formula. This process guarantees exploration of the entire design space while excluding models that have already been identified. This synthesis process exhaustively generates the design space, representing the full range of possible database configurations, encompassing all potential combinations and tradeoffs among design attributes.

To identify the actual Pareto-optimal subset of solutions for training purposes, the entire design space undergoes dynamic evaluation, resulting in values for each design attribute across all designs. This design space, along with its associated attribute values, is referred to as the '*tradespace*.' More specifically, it performs dynamic analysis, measuring performance based on parameters like insertion time, query selection time, and memory usage. In particular, it generates dynamic test suites for each of these designs. Each test suite consists of two components: (a) The SQL statements necessary to create a database that matches the specific design being tested; (b) A set of SQL statements that serve as a test load, designed to evaluate the insertion time, retrieval time, and storage space requirements for that particular database design. These three metrics are meticulously measured for each design, effectively defining the 'tradespace' and associating each design in the design space with corresponding values for each design attribute.

### 3.2 Exploring Design Space to Identify Superior Designs

In the subsequent phase, it navigates the tradespace, aiming to locate the Pareto frontier defined by designs that excel in various metrics. These Pareto-optimal designs are classified as preferred, while the remainder is designated as non-preferred. Nonetheless, relying solely on Pareto-optimal designs for the preferred category presents a substantial class imbalance issue, potentially undermining the learning model's effectiveness. Specifically, in practical scenarios, the number of dominated

design solutions (i.e., inferior solutions) often largely surpasses the number of superior design solutions (i.e., Pareto-optimal solutions), leading to class imbalance issue. Take, for instance, the E-commerce system outlined in Section 2. Figure 1 visualizes the distribution of dominated and Pareto-optimal design solutions within a real E-commerce design tradespace. Pareto-optimal solutions are denoted by triangles, while other dominated solutions are represented by black dots. The figure clearly illustrates that the number of superior design solutions is significantly smaller, constituting approximately 4% of the total, compared to the number of inferior design solutions.

Data augmentation has become a widely adopted technique in machine learning to combat data imbalance during model training. It involves applying various transformations or modifications to existing data, such as rotation, cropping, flipping, or adding noise. The objective of data augmentation is to generate new training examples that closely resemble the original ones but exhibit slight variations. This process enhances the overall performance and generalization capacity of machine learning models by providing them with a more diverse and resilient training dataset. However, in the context of tradespace analysis, where the solution space already comprehensively covers all potential design solutions, generating additional valid designs is not feasible. Previous studies have demonstrated that resampling techniques outperform undersampling methods, as the latter leads to the loss of valuable information [22]. Therefore, based on the aforementioned analysis and to rectify this imbalance, we employ minority resampling. This strategy involves augmenting the training data by replicating examples from the minority class.

Since the optimal sampling distribution is unknown in our problem, we empirically determine it by creating seven datasets with varying sampling distributions. These datasets involve successively oversampling from 20% to 80% of instances from class P (Pareto-optimal design solutions) with a 10% increment at each step. An optimal model in this context achieves higher recall for class P and higher precision for class NP (non-Pareto-optimal design solutions). In our learning-powered database design tradespace analysis, it is crucial to limit the omission of Pareto-optimal solutions.

### 3.3 Transformer-based DL Approach for Design Space Analysis

We conceptualize the design space analysis, particularly the determination of Pareto optimality, as an NLP problem. Our primary goal is to assess the quality of a design space using natural language text sequences representing triplets composed of object-oriented data models, relational database schemas, and object-relational mapping design strategies. To accomplish this, we employ a Transformer-based DL approach, leveraging its ability to process sequential data, including natural language text sequences [60]. Given the array of Transformer architectures and learning algorithms available, our focus is on identifying the most effective method for design space analysis.

Our exploration of Transformer models revolves around two primary dimensions: the incorporation (or absence) of prior knowledge and architectural diversity. A Transformer DL model acquires prior knowledge about linguistic patterns and semantics through parameter adjustments during extensive training on large natural language text corpora, such as Wikipedia [60]. These pre-trained models, commonly known as language models (LMs), excel in predicting missing text within training data. Recent Transformer-based pre-trained LMs have exhibited impressive general knowledge and reasoning capabilities [41], [59], [45], [46], [12], [65], [10], [16], demonstrating versatility not only in traditional natural language processing tasks but also in diverse domain tasks when fine-tuned for specific objectives [24], [45].

Our research aims to compare the effectiveness of fine-tuning pre-trained LMs against training non-LM Transformer models that lack prior knowledge. Unlike pre-trained LMs, which leverage extensive training on large corpora, non-LM Transformer models start with random weights and lack the inherent understanding of linguistic patterns present in pre-trained models. This

comparison enables us to evaluate the impact of prior knowledge on model performance and effectiveness across various tasks. Furthermore, our objective extends to identifying the most effective Transformer architecture, encompassing both encoder-decoder and encoder-only models.

To explore the diverse combinations of prior knowledge and architectural diversity in our Transformer models, we structure our domain data into natural language text sequences incorporating the triplet of object-oriented data models, relational database schemas, and object-relational mapping design strategies. We begin by fine-tuning pre-trained LMs with natural language text sequences. In contrast, for training non-LM Transformers, we adopt a dual approach. Initially, we utilize natural language text sequences. However, recognizing the challenge posed by the large variability of text tokens stemming from problem-specific definitions of object models and database schemas, we hypothesize (**Hypothesis 1**) that non-LMs may not effectively learn from such sequences. To mitigate this, we also employ standardized natural language text sequences for training non-LM Transformers. Standardization involves replacing user-defined identifiers with reusable identifiers, thereby reducing text token variability and aiming to enhance the learning process for these models.

An important consideration is whether the problem should be modeled as a classification task, where encoder-only BERT-type models predict a class token, or as a language/text generation task, where classification is reformulated as text generation by appending the class tag to the output sequence. The extent to which the prior knowledge and linguistic capability of pretrained large language models (LLMs) contribute to classification tasks depends on whether a pretrained LLM can infer the class tag (P/NP) in a zero-shot setting or if fine-tuning provides a significant advantage [43] [35] [46] [12].

By addressing these factors, we aim to gain deeper insights into the role of pretrained language models in structured classification tasks and their adaptability to problem-specific text sequences.

*3.3.1 Problem Formulation.* We present a formal description of the learning problem for encoder-decoder, encoder-only, and decoder-only models.

**Encoder-Decoder Transformer.** An encoder-decoder model transforms a space-separated lexical token-based input sequence $X \in (x_1, x_2, ..., x_l)$ into an output sequence $\hat{Y} \in (\hat{y}_1, \hat{y}_2, ..., \hat{y}_m)$. The model is trained via the backpropagation algorithm to minimize the loss between the predicted output sequence $\hat{Y}$ and the true output sequence $Y$.

In our problem, $X$ represents the triplet of object-oriented data models, relational database schemas, and object-relational mapping design strategies, while $Y$ represents the same triplet with an additional token representing the quality of the design (i.e., Pareto-optimal or not). Our hypothesis (**Hypothesis 2**) is that by constraining the model to reproduce the input sequence, we enhance its ability to comprehend the design solution effectively.

The encoder-decoder Transformer [60] employs an attention mechanism [8] to learn a context-aware representation of the input tokens. It consists of two architectural components: an encoder and a decoder. The encoder $f_E(\cdot)$ maps the input sequence $(x_1, x_2, ..., x_l)$ to an intermediate latent embedding sequence $(z_1, z_2, ..., z_l)$, where $\theta_E$ represents the encoder weights.

$$z = f_E(x_1, x_2, ..., x_l; \theta_E) \tag{1}$$

The decoder $f_D(\cdot)$ utilizes the latent embeddings $(z_1, z_2, ..., z_l)$ to generate an output sequence $(\hat{y}_1, \hat{y}_2, ..., \hat{y}_m)$ in an auto-regressive manner. During prediction, it employs a neural attention mechanism to identify the tokens in the encoded input sentence $z$ most relevant to the target token it is currently predicting. The probability of generating the $m$-th token $\hat{y}_m$ is given by

$$p(\hat{y}_m | \hat{y}_{<m}; z_1, z_2, ..., z_l) = \text{softmax}(f_D(\hat{y}_{<m}; z_1, z_2, ..., z_l; \theta_D)) \tag{2}$$

where $\theta_D$ denotes the decoder weights. For training an encoder-decoder model, the multi-class cross-entropy loss function is utilized. The number of classes in the loss function is determined by the total number of tokens in the vocabulary. For a batch size $B$, the loss function is defined as:

$$\mathcal{L} = -\sum_{b=1}^{B}\sum_{m=1}^{M} y_m^b \log \hat{y}_m^b \tag{3}$$

**Encoder-only Transformer.** In contrast to the encoder-decoder model used for token generation, the encoder-only model is employed solely for learning a representation of the input sequence, thus eliminating the need for a decoder. Similar to the encoder in the encoder-decoder model, the encoder-only Transformer creates a contextual representation $Z$ of the input sequence $X$ using the attention mechanism [8]. During inference, the encoded representations $Z$ can be utilized for various downstream tasks, such as text classification or named entity recognition.

Here, the input sequence is defined as $X \in (x_1, x_2, \ldots, x_l)$, where $X$ represents the triplet of object-oriented data models, relational database schemas, and ORM design strategies. The output sequence is given by $\hat{Y} \in (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m)$, where $\hat{Y}$ consists of only the additional token representing the design quality, without reconstructing the original input. This formulation makes encoder-only models particularly suitable for classification-based tasks.

We utilize the encoder-only model as a text classifier to determine the binary label (Pareto-optimal or not) of an input sequence representing the triplet. We apply the softmax function to the final representation followed by a cross-entropy loss function.

$$p(y_k|X) = \text{softmax}(f_{\text{classifier}}(z; \theta_{\text{classifier}})) \tag{4}$$

where $y_k$ represents the class probabilities, $\theta_{\text{classifier}}$ denotes the classifier parameters, and $f_{\text{classifier}}(\cdot)$ is the classifier function.

The cross-entropy loss for training an encoder-only model with batch size $B$ is defined as:

$$\mathcal{L} = -\sum_{b=1}^{B}\sum_{k=1}^{K} y_k^b \log(p(y_k^b|X^b)) \tag{5}$$

where $K$ is the number of classes, $y_k^b$ is the true label for the $b$-th example, and $p(y_k^b|X^b)$ is the predicted probability of class $k$ for the $b$-th example.

**Decoder-Only Transformer.** A decoder-only Transformer generates an output sequence in an autoregressive manner, modeling the probability distribution of the next token given the sequence of preceding tokens. Unlike the encoder-decoder model, it does not employ a separate encoder module; instead, the decoder itself processes the input. The input sequence is defined as $X \in (x_1, x_2, \ldots, x_l)$, where $X$ represents a triplet consisting of object-oriented data models, relational database schemas, and ORM design strategies. The model generates an output sequence $\hat{Y} \in (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m)$, which reconstructs the input triplet and appends an additional token indicating design quality.

The decoder-only Transformer uses an attention mechanism [8] to focus on relevant portions of the input sequence while generating each token in the output. At each decoding step, the probability of generating the $m$-th token $\hat{y}_m$ is determined by:

$$p(\hat{y}_m|\hat{y}_{<m}; X) = \text{softmax}(f_D(\hat{y}_{<m}; X; \theta_D)), \tag{6}$$

where $f_D(\cdot)$ is the decoder function, $\hat{y}_{<m}$ represents the sequence of previously generated tokens, and $\theta_D$ denotes the weights of the decoder.

Unlike the encoder-decoder model, the decoder-only Transformer leverages self-attention to compute the representations of both the input tokens and the intermediate outputs during generation. This architecture eliminates the need for a separate encoder, reducing computational overhead while maintaining contextual awareness.

The cross-entropy loss function to minimize the difference between the predicted sequence $\hat{Y}$ and the true sequence $Y$. The loss function for a batch size $B$ is defined as:

$$\mathcal{L} = -\sum_{b=1}^{B}\sum_{m=1}^{M} y_m^b \log(\hat{y}_m^b), \tag{7}$$

where $y_m^b$ is the true label and $\hat{y}_m^b$ the predicted probability for the $m$-th token in the $b$-th example.

### 3.4 Data Preparation for Transformer Models

To structure our domain data into natural language text sequences, suitable for training and fine-tuning the Transformer models, we employ a systematic approach based on the input triplet $\langle O, D, S\rangle$, where $O$ represents object-oriented data models, $D$ represents relational database schemas, and $S$ represents object-relational mapping (ORM) design strategies.

The process of converting this triplet into text sequences involves the following steps: We perform the following steps: *(1) Tokenization:* We tokenize each component of the triplet ($O$, $D$, and $S$) into a series of lexical tokens that represent the structure and elements of the models, schemas, and ORM strategies. *(2) Sequence Construction:* We concatenate these tokens into a single space-separated sequence, maintaining the order of $O$, $D$, and $S$. This sequence forms our natural language text representation. *(3) Standardization* (for standardized natural language text only): We replace user-defined identifiers with expert-defined reusable identifiers to reduce variability in the token space. We utilize two types of text sequences: *(1) Natural language text:* This retains the original tokens from the $\langle O, D, S\rangle$ triplet. *(2) Standardized natural language text:* This replaces user-defined identifiers with standardized ones.

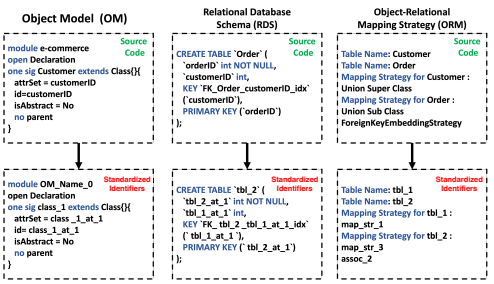


Fig. 3. Standardization: Transforming the user-defined identifiers in the OM, RDS, and ORM strategy by reusable standardized identifiers.

The standardization process involves transforming three lexical data structures: OM, RDS, and the ORM strategy into vectors of space-separated lexical tokens. During this transformation, we utilize reusable identifiers to replace user-defined entities (e.g., module/class names, types, variable names, and string literals) from the data structures.

Figure 3 illustrates this process. The user-defined module identifier `e-commerce` in the OM source code is replaced by `OM_name_0` in the standardized version. Similarly, in the OM source code, the user-defined class name `Customer` is replaced in the standardized OM by `class_1`. We append the general term `type` to the replaced identifier while representing common data types (class or variable), such as Integer, String, and Boolean. The class attributes are transformed in a similar fashion. Standardization of the RDS and ORM follows the same process. For example, user-defined schema and table names of the original RDS code are replaced by standardized identifiers. This standardization process helps reduce learning complexity while maintaining the syntactic structure of the original models [21].

For encoder-decoder and decoder-only models, we append a label ($P$ or $NP$) to the input sequence $X$ to create the output sequence $Y$. For encoder-only models, we use the same input sequence $X$, with a binary label $Y \in \{0, 1\}$ representing $NP$ and $P$, respectively. This structured approach ensures that our domain data are consistently transformed into text sequences suitable for processing by various Transformer architectures, while maintaining the essential information from the original object-oriented data models, relational database schemas, and ORM strategies.

## 4 EXPERIMENTAL EVALUATION

In this section, we focus on evaluating the performance of DesignTradeoffSculptor by answering the following research questions:

- **RQ1.** Does tradespace analysis enable identification of Pareto-optimal design schemas that are entirely overlooked by the state of the practice ORM design tools?
- **RQ2.** How effectively does our learning-based approach identify Pareto-optimal solutions compared to state-of-the-art specification-driven tradespace analysis techniques?
- **RQ3.** What is the performance improvement achieved by our learning-based approach compared to state-of-the-art in tradespace analysis?
- **RQ4.** Which Transformer-based approach most effectively identifies Pareto-optimal solutions?

**Tool Implementation.** We have implemented our approach in a tool called DesignTradeoffSculptor. We made research artifacts and experimental data available [1]. The specifications were created in Alloy [29] and executed with Trademaker [7], the Java API of Alloy 5, along with the Kodkod model finder [58], which powers that particular version of the Alloy Analyzer. Additionally, the SAT4J SAT solver [9] was employed. The translation of the input models into the Alloy language is implemented using Apache FreeMarker template engine [3]. We have implemented DesignTradeoffSculptor 's data abstraction in Python 3. Notably, the prototype implementation of DesignTradeoffSculptor doesn't require end-users, such as software developers and database designers, to possess formal specification or machine learning knowledge.

**Transformer Models.** Our study incorporated both language model (LM) and non-LM Transformer architectures. We employed T5 (Text-to-Text Transfer Transformer) [45] and CodeT5 [64] as our encoder-decoder pre-trained LMs. T5 was chosen for its versatility and exceptional performance across various natural language processing tasks, with a context window of 512 tokens. CodeT5, which follows the architecture of T5, was specifically utilized for its pre-training on code-related tasks. We utilized the Large variants of both T5 and CodeT5 models, each equipped with 770 million pre-trained parameters.

For the encoder-only pre-trained LM, we utilized both BERT (Bidirectional Encoder Representations from Transformers) and CodeBERT. We selected BERT for its ability to capture bidirectional context and semantic representations, making it ideal for efficient textual information encoding[16]. Specifically, we experimented with the BERT base model, which contains 110 million pre-trained parameters. Additionally, we incorporated the CodeBERT base model, which extends BERT's architecture to enhance code understanding and representation learning[19]. Pre-trained on both natural and programming language data, CodeBERT comprises 125 million parameters, offering specialized capabilities for code-related tasks.

For our decoder model, we employed Llama [59], specifically Llama 3.1 [37], which has 8 billion parameters. For the non-LM Transformer, we implemented a bespoke encoder-decoder model with attention mechanisms for context-aware token representation. This model comprises 20,205,976 trainable parameters, initialized with random weights. Both the encoder and decoder consist of a single layer. We trained the model using a dataset of 13,269 instances, incorporating both natural language text sequences and their standardized versions.

**Deep Learning Experimental Setup.** Language Models T5, CodeT5, BERT, and CodeBERT were fine-tuned using the AdamW optimizer [34] with a learning rate of $2 \times 10^{-5}$. For T5 and CodeT5 models, experiments used a batch size of 4 over 12 epochs. BERT and CodeBERT models used a batch size of 16 over 5 epochs. Batch sizes were constrained by memory limitations during fine-tuning. The Llama 3.1 model was fine-tuned using AdamW with a learning rate of $2 \times 10^{-4}$.

Experiments used a training batch size of 1 and an evaluation batch size of 2, over 8 epochs. Gradient accumulation with six steps simulated larger batch sizes, mitigating memory constraints.

The non-LM encoder-decoder Transformer underwent training for 120 epochs. The selection of epoch numbers was guided by empirical considerations, including training accuracy, loss, validation accuracy, and loss. We experimented with various optimizer choices and ultimately selected the Adam optimizer [32] due to its superior performance. The hyperparameters employed for the Transformer model include a learning rate of 0.001, weight decay of $1.0 \times 10^{-5}$, a batch size of 64, a vocabulary size of 15,000, and a sequence length of 500. We utilized NVIDIA A100 and NVIDIA Tesla V100 GPUs for distributed training, completing the fine-tuning (LM) and training (non-LM) processes for each model in under an hour.

**Subject Systems.** Our experimental subjects encompass nine distinct database-driven software systems. Five of these subject systems were directly sourced from prior studies [6, 7, 33], while the other four were adapted from publicly-available open-source systems. These selected subjects offer a diverse range of system sizes and complexities, carefully matched to the scales used for evaluating state-of-the-art tradespace analysis techniques.

The five systems obtained from prior studies [6, 7, 33] are as follows: (1) A software system developed to support decision-making through design space exploration. (2) an E-commerce system adopted from Lau and Czarnecki [33]. It represents a common architecture for open-source and commercial E-commerce systems. (3) A cyber-social operating system (CSOS) intended to aid in coordinating people and tasks. (4) A straightforward CRM system designed for tracking customers and orders. (5) Flagship, a system created for tracking revisions to shared documents.

Additionally, we incorporated four more systems from publicly-available open-source systems into our study: (1) A Bank Management System that handles customer data, accounts, and loans [25]. (2) A Merit Badge Tracker that manages badge awards and camping activities [56]. (3) A streamlined Online Store System with core e-commerce functionality [40]. (4) A Library Management System that coordinates book loans and library operations [14].

**Baselines and Measures.** To address RQ1, we used four widely-used object-relational database mapping techniques, namely SQL-Alchemy [48], Hibernate [18], Entity Framework [38], and Django ORM [17], as our baseline. These techniques represent the current state-of-the-practice in ORM database design. We synthesized and evaluated solutions for our target object data models, splitting the dataset into a training set (85%) and a testing set (15%) to train and evaluate our transformer model. Within the 85% training data, we further allocated a portion for validation, resulting in 70% for training, 15% for validation, and 15% for testing. This three-way split structure allowed us to provide the model with sufficient data for learning while preventing overfitting. We used the validation set for real-time performance monitoring, early stopping, and hyperparameter tuning. To further mitigate overfitting, we employed appropriate regularization techniques and hyperparameter optimization using the validation data. The separate test set was maintained for the final, unbiased evaluation of the model's performance. This approach promises that our model generalized well to unseen data while maintaining robust performance. Furthermore, we conducted nine other experiments, each involving training the transformer model on the tradespace of eight subject systems and testing it on the remaining subject system not included in the training set.

To address RQ2, we employed several evaluation metrics. The first metric is Generational Distance (GD) [67], which assesses the distance between the Pareto front generated by DesignTradeoffSculptor, the baselines, and a reference frame. It measures the closeness of the designs produced by each technique to the true Pareto frontier, which represents the set of non-dominated (optimal) ORM designs. In GD calculation, for each design identified by our approach and the baseline

techniques, the distance to the nearest point on the Pareto frontier is computed. The GD value is then determined as the average of these distances. Smaller GD values indicate that the designs are closer to the Pareto frontier and, thus, are of higher quality in terms of Pareto optimality. The second metric is Hypervolume (HV) [68], which gauges the proximity of our obtained Pareto front to a reference frame and the diversity of its points. A higher diversity is considered better. HV measures the volume enclosed by these points. We compare the hypervolume of the Pareto front estimated by our ML-based approach with the hypervolume of the solution obtained using all four leading ORM design techniques.

To address RQ3, we adopted the original version of TradeMaker [7] as our baseline, representing the current state-of-the-art in tradespace analysis for database design. We quantified the total time, in seconds, and memory usage required by TradeMaker to conduct dynamic analysis for tradespace computation and identify the Pareto frontier for each subject system. We then compared these time and memory usage metrics with those taken by DesignTradeoffSculptor for identifying the Pareto frontier for the same set of subject systems.

To address RQ4, we conducted experiments with both LM and non-LM models using two types of datasets $(X, Y)$: natural language text sequences and standardized natural language text sequences, respectively (refer to Section 3.4). These datasets were constructed from nine database-driven software applications. The primary challenge with these datasets is their significant skew towards $NP$ instances. This skew is attributed to three large applications (E-commerce, Library Management, and CSOS), where the number of Pareto-optimal ORM strategies is notably smaller. To ensure a balanced distribution of $P$ and $NP$ instances, we utilized the random sampling technique [22], involving undersampling from the majority class and oversampling from the minority class [23].

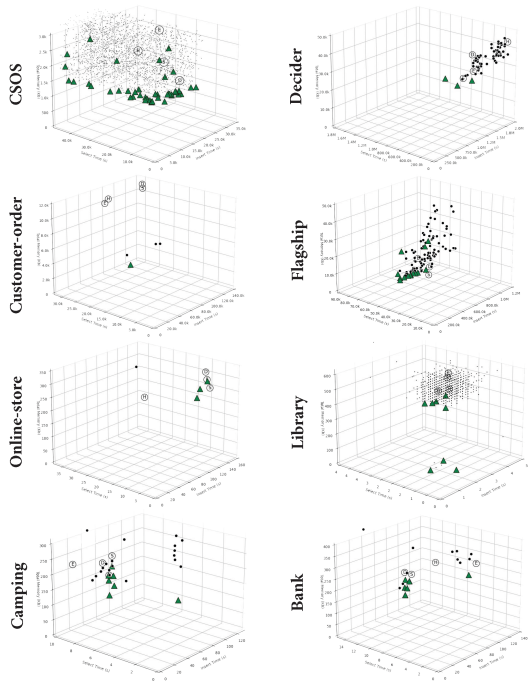


Fig. 4. Tradespace (Dynamic Tradeoff analysis) for the subject systems' database designs. Each black dot represents the performance of a design schema. Design solutions generated by state-of-the-practice object-relational mapping tools, namely SQLAlchemy [48], Hibernate [18], Entity Framework [38], and Django ORM [17], are represented respectively by Ⓢ, Ⓗ, Ⓔ, and Ⓓ. The designs produced by widely-used state-of-the-practice tools are noticeably far from the Pareto-optimal solutions, denoted by triangles, in terms of performance.

For an unbiased evaluation of the Transformer models, we ensure that each model is trained or fine-tuned using instances from eight software systems and subsequently tested using the data from the remaining system. This process entails conducting nine experiments for each model, where it is trained on data from eight systems and then evaluated on the remaining application. In each iteration, a new test set from the remaining system is utilized to assess the model's performance.

In the analysis of design tradeoffs, we focus on two crucial metrics: precision for class $NP$ and recall for class $P$. Recall assesses the model's ability to correctly identify instances of a particular

class, while precision measures the accuracy of such identifications. We consider a model to be optimal if it achieves higher precision for class *NP* and higher recall for class *P*. A perfect recall for class *P* (i.e., 1) indicates that the model accurately identifies all instances of class *P*. Conversely, perfect precision for class *NP* (i.e., 1) suggests that the model does not classify any *P* instances as *NP*, resulting in zero false positives.

### 4.1 Results for RQ1: Impacts of Tradespace Analysis

Figure 4 illustrates tradeoff space plots for eight distinct subject systems, complementing the one showcased in Figure 1 for the E-commerce system. These plots project the results into 3-D tradeoff plots. Each plot illustrates tradeoff information in three dimensions: insertion time, retrieval time, and storage space consumption. In these plots, each dot represents the analysis result of one database design. The hollow triangles represent Pareto-optimal designs identified through tradespace analysis in their respective tradeoff spaces, showcasing the Pareto-optimal solutions in each plot. To assess the effectiveness of tradespace analysis in discovering superior designs, we compare the results with those produced by widely-used industrial database design tools. The design solutions generated by state-of-the-practice object-relational mapping tools, namely SQLAlchemy [48], Hibernate [18], Entity Framework [38], and Django ORM [17], are respectively denoted as Ⓢ, Ⓗ, Ⓔ, and Ⓓ. These tools, in contrast to our approach which relies on tradespace analysis, follow a single-point strategy and produce only a single design. This limited approach fails to offer decision-makers comprehensive insights into available design solutions or the tradeoffs associated with time and space performance.

The results in Figure 4, reveal that out of the 128 Pareto-optimal solutions generated across the nine systems, the state-of-the-practice tools were only able to produce one in the online store experiment. For the other eight experiments, these tools failed to generate any of the Pareto-optimal solutions. The design solutions generated by the baseline tools are distant from the Pareto fronts.

> The results indicate that systematic tradespace analysis enables uncovering superior design solutions that are often overlooked by the existing techniques widely used by developers.

### 4.2 Results for RQ2: DesignTradeoffSculptor in Practice

To assess the effectiveness and accuracy of our learning-powered technique in identifying Pareto-optimal database designs, we employed two Pareto front quality metrics, i.e., Generational Distance (GD) and Hypervolume (HV). These metrics were used to compare the quality of DesignTradeoffSculptor's design solutions with those generated by widely-used baseline tools across nine subject systems.

Table 1. Comparative Generational Distance Analysis of DesignTradeoffSculptor Against Popular ORMs: Assessing Pareto Frontier Proximity

| Object Model | Tradeoff Sculptor | Hibernate | Django | SQL Alchemy | Entity Fwk. |
|---|---|---|---|---|---|
| Customer-Order | **0.02** | 0.54 | 0.71 | 0.69 | 0.50 |
| Online Store | **0.00** | 0.73 | 0.15 | **0.00** | 0.03 |
| Bank | **0.01** | 0.18 | 0.09 | 0.09 | 0.13 |
| Camping | 0.06 | **0.00** | 0.07 | 0.11 | 0.24 |
| Flagship | **0.01** | **0.01** | 0.02 | 0.12 | 0.13 |
| Decider | **0.00** | 0.48 | 0.27 | 0.16 | 0.45 |
| Library Mgmt. | **0.01** | **0.01** | 0.03 | 0.08 | 0.04 |
| CSOS | **0.03** | 0.34 | 0.17 | 0.08 | 0.42 |
| E-commerce | **0.06** | 0.33 | 0.64 | 0.37 | 0.58 |
| **Average** | **0.02** | 0.29 | 0.24 | 0.19 | 0.28 |

Table 1 presents the results of a comparative Generational Distance (GD) analysis [67], which assesses the proximity of DesignTradeoffSculptor's solutions to the Pareto frontier across nine subject systems. We compared DesignTradeoffSculptor with four state-of-the-art, commonly used database design tools. The GD metric measures the average Euclidean distance between the solutions generated by each technique and the true Pareto frontier. A smaller GD value indicates closer proximity to the Pareto frontier, representing better performance in terms of Pareto optimality.

DesignTradeoffSculptor achieves an average GD of 0.02 across the nine subject systems, indicating that its solutions closely approximate the Pareto frontier. In contrast, Hibernate, Django, SQLAlchemy, and Entity Framework have average GD results ranging from 0.19 to 0.29. This suggests that their solutions deviate more significantly from the optimal trade-offs represented by the Pareto frontier. While the absolute GD values do not translate directly to specific percentages of optimality, they provide a relative measure of each tool's ability to generate Pareto-optimal designs. The lower GD value of DesignTradeoffSculptor implies that it consistently recommends designs that are very close to the Pareto frontier compared to the other tools evaluated.

The heatmap in Figure 5 illustrates the results of a comparative analysis using the Hypervolume indicator [68], which measures the volume of the portion of the design space that is dominated by the solutions produced by each technique across nine subject systems. We compared DesignTradeoffSculptor with four state-of-the-art, commonly-used database design tools. The HV results offer insights into the performance of each technique in terms of Pareto dominance and coverage of the design space. Specifically, it calculates the space covered by the solutions of each approach. A higher Hypervolume value signifies superior Pareto dominance. The approach with the higher hypervolume demonstrates that the solutions in the set cover a larger portion of the design space while remaining as close as possible to the Pareto frontier. The average Hypervolume result of 0.85 across nine subject systems for DesignTradeoffSculptor indicates that it dominates a substantial portion of the design space while maintaining proximity to the Pareto frontier. In contrast, the average Hypervolume results for Hibernate (0.22), Django (0.12), SQLAlchemy (0.27), and Entity Framework (0.10) suggest that these techniques cover significantly smaller portions of the design space and do not perform as well in terms of Pareto dominance. According to the results, the solutions produced by existing Database design tools are less diverse and do not reach as far towards the Pareto frontier, suggesting that they may not perform as effectively in finding a wide range of design solutions that balance various design attributes.

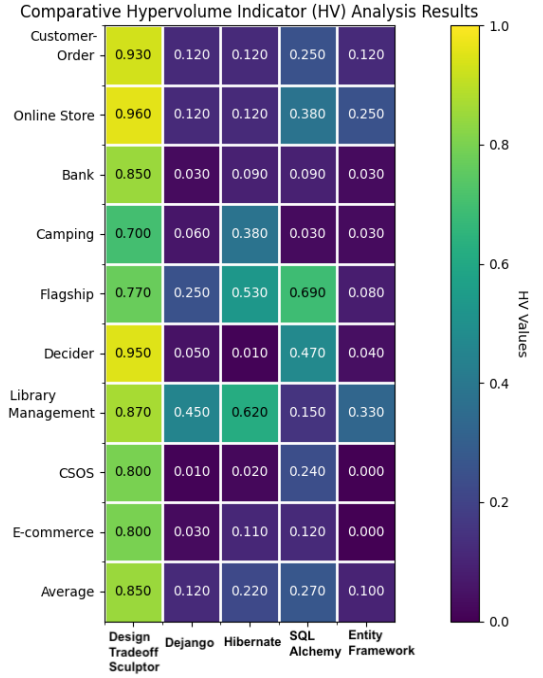


Fig. 5. Comparative Hypervolume Indicator Heatmap of DesignTradeoffSculptor Against SQLAlchemy [48], Hibernate [18], Django [17], and Entity Framework [38].

We include HV measurements for all approaches, including baseline ORM tools that generate single-point solutions, to ensure a consistent comparison. For single-point solutions, HV quantifies

the solution's dominance in the multi-objective space, calculated relative to a reference point representing the worst possible values for each objective. This application of HV to single points provides crucial information about a solution's overall quality across multiple criteria. Using HV uniformly allows direct comparison between single-point ORM methods and multi-point solutions from DesignTradeoffSculptor and Trademaker. This analysis showcases the advantages of exploring a broader ORM design solution space, evidencing the benefits of considering diverse solution sets.

> The data suggests that DesignTradeoffSculptor consistently outperforms common database design tools, as evidenced by smaller GD values and higher HV results, indicating closer proximity and greater diversity of solutions to the Pareto frontier in database design.

### 4.3 Results for RQ3: Performance Improvement

Table 2 depicts the results of our performance comparison between TradeMaker and DesignTradeoffSculptor. On average, TradeMaker takes approximately 1,313,707 seconds (about 15.2 days) for analysis, while our approach completes the analysis in just 1,097 seconds (about 18 minutes). This demonstrates a substantial reduction in analysis time. In terms of memory usage, TradeMaker consumes an average of 35.7 GB, while our approach for the same subject systems uses an average of around 8.2 MB.

Table 2. Time is measured in seconds and memory usage in KB for the comparison between TradeMaker [7] and DesignTradeoffSculptor for identifying Pareto frontiers.

| | Trademaker [7] | | DesignTradeoffSculptor | |
|---|---|---|---|---|
| **Object Model** | **Time** | **Memory** | **Time** | **Memory** |
| Customer-Order | 7,056 | 67,168 | 14 | 61 |
| Online Store | 1,004 | 10,589 | 24 | 31 |
| Decider | 63,465 | 601,680 | 20 | 134 |
| Flagship | 63,971 | 538,496 | 98 | 423 |
| E-Commerce | 5,697,328 | 156,952,992 | 6,054 | 40,800 |
| CSOS | 5,071,169 | 177,032,670 | 2,566 | 20,000 |
| Bank | 1,923 | 27,021 | 20 | 134 |
| Library Mgmt. | 915,642 | 1,600,742 | 1,055 | 13,700 |
| Camping | 1,808 | 30,208 | 24 | 118 |
| **Average** | 1,313,707 | 37,429,063 | 1,097 | 8,378 |

It is important to note that TradeMaker's performance is due to its exhaustive algorithm design rather than implementation issues. TradeMaker formally synthesizes the entire solution space, covering all Pareto-optimal solutions without approximation. Consequently, TradeMaker's Generational Distance (GD) is always 0.0, and its Hypervolume (HV) is always 1.0, representing the ground truth for these metrics. In comparison, DesignTradeoffSculptor achieves near-optimal results with substantially reduced computational resources. Our approach yields an average GD of 0.02 and an average HV of 0.85 across all test cases, demonstrating its ability to closely approximate the Pareto-optimal solutions identified by TradeMaker.

> This comparison illustrates the trade-off between exhaustive exploration and efficient approximation in tradespace analysis. TradeMaker finds all Pareto-optimal solutions but is computationally expensive. DesignTradeoffSculptor balances solution quality and efficiency, making it practical for real-world scenarios with time and resource constraints.

### 4.4 Results for RQ4: Identifying the Effective Transformer-based learning approach

Figure 6 illustrates the results obtained from experiments conducted with each Transformer model. To ensure reliability, each experiment was repeated three times. We report the average and standard deviation values of precision, recall, and accuracy for all experiments conducted with each model (nine experiments per model). Our findings indicate that both the decoder-only language model

(Llama 3.1) and the encoder-decoder non-LM model achieve superior performance compared to other Transformer-based models. The decoder-only model (Llama 3.1) achieved a recall of 0.84 for class *P* and a precision of 0.96 for class *NP*, leveraging its extensive prior knowledge and linguistic capabilities. The encoder-decoder non-LM model, trained from scratch with random initial weights, performed strongly when trained on standardized text, achieving a recall of 0.85 for class *P* and a precision of 0.96 for class *NP*. However, its performance declined when trained on raw data, with a recall of 0.75 for class *P* and a precision of 0.91 for class *NP*.

These results support Hypothesis 1 and Hypothesis 2 (refer to 3.3). Hypothesis 1, which posits that reducing variability in token space through standardized text improves model performance, is validated by the superior results of the encoder-decoder non-LM model on standardized data. Hypothesis 2, which suggests that structured input processing enhances comprehension of design solutions, is supported by the strong recall and precision demonstrated by both the encoder-decoder non-LM and decoder-only LM models.

General-purpose language models, such as T5 and BERT, and domain-specialized models, such as CodeT5 and CodeBERT, exhibit limitations in evaluating design quality due to challenges in balancing precision and recall when distinguishing Pareto-optimal (*P*) from non-optimal (*NP*) design alternatives. For example, T5 achieves an accuracy of 63% but suffers from low precision (0.20) for class *P*. Similarly, BERT achieves high recall for class *P* (0.77) but low recall for class *NP* (0.21), resulting in an overall accuracy of only 30%.

Domain-adapted models, like CodeT5 and CodeBERT, underperform in this context. CodeT5 shows 62% accuracy and 0.67 recall (0.15 precision) for class *P*, while CodeBERT achieves 30% accuracy with 0.28 recall for class *NP*. While effective for some code tasks, these models may lack depth for applications reasoning for assessing optimal design trade-offs.

Overall, these findings underscore the importance of selecting appropriate learning approaches tailored to the specific requirements of the task at hand, especially in complex domains like design space analysis. Furthermore, our results emphasize the benefits of employing an NLP approach for design space analysis and offer valuable insights for the development of more effective approaches using Transformer DL models for such tasks.

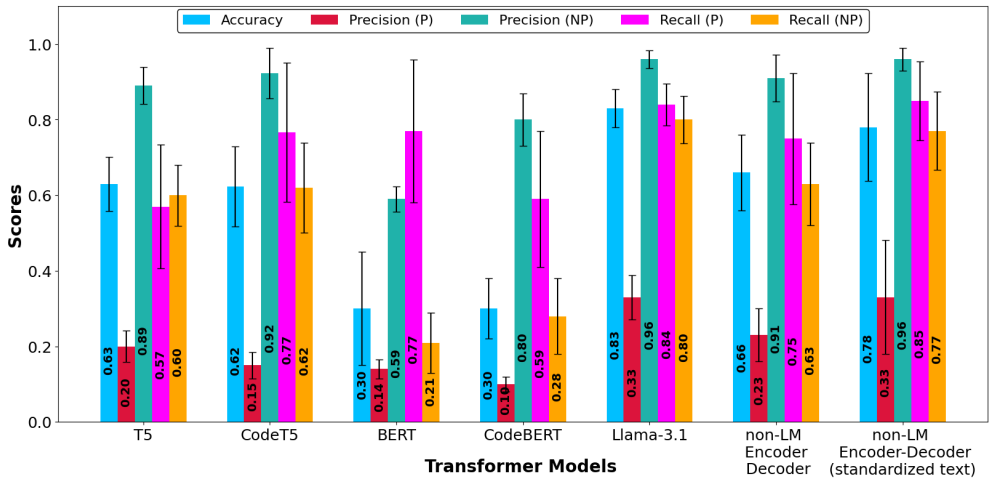


Fig. 6. Comparison of transformer models for Pareto-optimal solution recognition. Mean and standard deviation for precision, recall, and accuracy across nine experiments per model are reported.

The superior performance of the encoder-decoder non-LM model when trained on standardized data supports Hypothesis 1 (reducing token space variability improves performance), while the strong recall and precision demonstrated by both the encoder-decoder non-LM and decoder-only LM (Llama 3.1) models validate Hypothesis 2 (structured input processing enhances comprehension of design solutions).

## 5 DISCUSSION

*Comparative Performance of Non-LM and LM-based Models.* Our results demonstrate that both the encoder-decoder non-LM model and the decoder-only language model (Llama 3.1) achieve superior performance compared to other Transformer-based models in evaluating ORM design quality. The non-LM model's strong performance, particularly when trained on standardized text, can be attributed to its specialized architecture designed to capture structural relationships in ORM design spaces effectively. This aligns with our Hypothesis 1, which posits that reducing variability in token space through standardized text improves model performance. Similarly, Llama 3.1's success stems from its extensive pre-training on diverse corpora, including structured data and code, as well as its advanced decoder-only architecture with grouped query attention. Both models benefit from structured input processing, supporting Hypothesis 2, which suggests that such processing enhances comprehension of design solutions.

In contrast, general-purpose language models (T5, BERT) and domain-specialized models (CodeT5, CodeBERT) exhibit limitations in this task. These models struggle to balance precision and recall when distinguishing Pareto-optimal from non-optimal design alternatives, resulting in lower overall accuracy. For instance, T5 achieves 63% accuracy but suffers from low precision (0.20) for Pareto-optimal solutions, while BERT shows high recall (0.77) for Pareto-optimal solutions but low recall (0.21) for non-optimal ones. This underperformance can be attributed to their lack of task-specific adaptation and architectural limitations in capturing the nuanced relationships in ORM design spaces. Our findings underscore the importance of tailored architectures and appropriate input processing in complex domains like design space analysis, highlighting the potential of both specialized non-LM approaches and advanced, fine-tuned language models like Llama 3.1 in tackling such tasks.

*Generalization to Domain-Dependent Design Spaces.* While our study focused on ORM problems without domain-specific naming dependencies, we recognize the need to generalize our approach to design-space exploration that may require domain knowledge. In scenarios where quality attributes are naming-dependent, LM-based models might have an advantage due to their natural language understanding. To address this, we propose future research directions, including developing hybrid models that combine non-LM and LM strengths, such as incorporating domain-specific embeddings into non-LM models or using a two-stage approach where LM models interpret domain-specific terminology. Comparative studies across diverse domains would further elucidate the relative strengths of non-LM and LM-based models in such scenarios.

*Practical Application Workflow.* Our proposed method can be integrated into various stages of the software development lifecycle, particularly during initial database design and performance optimization phases. The following workflow outlines how developers can apply our approach to achieve optimal ORM designs: Developers begin by inputting their initial ORM design or a set of design alternatives into our tool. This can be done during the early stages of database design or when refactoring an existing system for performance improvements. Our trained Transformer

model then evaluates the input design(s), considering multiple quality attributes such as query performance, data integrity, and maintainability. The model generates a set of Pareto-optimal design recommendations, highlighting trade-offs between different quality attributes. Developers review these recommendations, taking into account project-specific constraints and requirements that may not be captured by the model. Based on the model's feedback and their own expertise, developers refine the ORM design. This step may involve selecting one of the recommended designs or making further adjustments. This process can be repeated iteratively to fine-tune the design further.

To facilitate adoption, we are developing a user-friendly tool that implements this workflow, allowing developers to easily input their designs, receive recommendations, and visualize trade-offs between different quality attributes. This tool will help bridge the gap between theoretical optimality and practical application in real-world software engineering contexts.

## 6 RELATED WORK

In this section, we provide overview of relevant research and examine them in light of our research.

**Design space exploration.** A relevant thrust of research has focused on design space exploration [31, 39, 44, 50, 54]. Kang, Jackson, and Schulte [31] used formal methods and an SMT solver for exploration, focusing on identifying satisfying models rather than optimal ones. Recent efforts have aimed at constraining the model space during exploration. Sullivan et al. [54] introduced a method to synthesize models satisfying a specified abstraction function. Porncharoenwase et al. [44] proposed an approach to identify a representative sample covering a broad syntactic range. Nelson et al. [39] developed Aluminum, allowing manual user guidance in exploration. These approaches primarily aim to reduce the model space rather than efficiently identifying optimal models. In contrast, DesignTradeoffSculptor makes no such assumptions, offering significant speedups in identifying Pareto-optimal designs while fully exploring the entire design space.

**Tradeoff analysis.** Tradeoff analysis is another relevant research area [2, 5, 11, 42, 47, 55]. Bondarev et al. [11] introduced DeepCompass for analyzing architectural alternatives, but it requires manual specification and lacks architecture synthesis support. Petke et al. [42] used genetic improvement to transplant code between system versions for performance enhancement. Schulte et al. [47] optimized non-functional properties unaddressed by compilers. Aleti et al. [2] developed ArcheOpterix to optimize embedded system architectures using an evolutionary algorithm on AADL models, focusing on component deployment. Martens et al. [36] proposed PerOpteryx to enhance initial software architectural models by searching for Pareto-optimal solutions using genetic algorithms with Palladio Component Models. Unlike our method of automatically identifying Pareto frontiers from input models, these research efforts often start from a specific solution point and incrementally optimize it. We view these two approaches to tradeoff analysis as complementary.

**Neural Machine Translation.** Neural Machine Translation (NMT) has been applied to various software engineering problems [4, 20, 21, 30, 63]. Cerebro [21] used NMT for mutation testing optimization, showing strong inter-project prediction but limited by a low subsuming mutant recall rate. It employed RNN-based NMT, which underperforms compared to Transformer models for certain tasks. Wang et al. [63] explored Seq2Seq pretraining for NMT, finding improvements but introducing discrepancies affecting quality. CURE [30] leveraged pre-training techniques for neural program repair, facing challenges in benchmarking due to data variations and Java dependency. AutoTransform [57], an NMT-based approach for automating code transformations in code reviews, has limitations in scope and scalability. Gad et al. [20] introduced a transformer-based model for pseudo-code generation, showing promise with Python data but limited by single-language focus. Unlike all prior work, our approach, to the best of our knowledge, is the first to employ NMT

extensively for identifying Pareto optimal solutions in tradeoff selection, reducing reliance on resource-intensive tradespace dynamic evaluation and saving time and memory.

## 7 THREATS TO VALIDITY

External validity threats relate to the generalizability of our findings beyond the specific software database systems used in our experiments. While DesignTradeoffSculptor has demonstrated its success across a diverse set of systems, it is important to acknowledge that its performance may vary when applied to systems from different domains or with distinct characteristics. To mitigate this issue and enhance the external validity of our study, we carefully selected a varied set of real-world database-driven software systems from various application domains, encompassing more than 13,000 database designs across nine subject systems.

The primary internal validity concern in our experimental evaluation pertains to the accuracy and reliability of our custom implementation. To mitigate this threat, we rigorously debugged and extensively tested our code, including the implementation of unit tests. These measures ensure that our custom implementation accurately analyzes database models, reducing the risk of implementation-related errors influencing our results.

Construct validity threats typically revolve around the selection of appropriate metrics and measures for assessing the phenomenon under investigation. In this study, we have taken significant measures to address construct validity concerns. Our choice of metrics, including Generational Distance and Hypervolume, aligns with established practices for evaluating tradeoff analysis techniques. Furthermore, our use of diverse subject systems and comprehensive training and validation of the transformer model adds to the robustness of our approach, minimizing potential validity threats.

## 8 CONCLUSION

This paper introduces DesignTradeoffSculptor, a novel machine learning-based solution for swiftly identifying optimal tradeoffs in database designs for object-oriented software systems. By training a transformer model on a dataset of formally analyzed database designs, DesignTradeoffSculptor efficiently pinpoints Pareto-optimal design tradeoffs, significantly reducing the need for costly design tradespace analysis. Evaluation across various software database systems showcases DesignTradeoffSculptor's ability to unveil previously overlooked Pareto-optimal design schemas while drastically improving analysis efficiency, with an average time reduction exceeding 98.21% compared to current state-of-the-art methods.

Future work could explore extending it to other domains like software architecture or code optimization, promising valuable insights. Additionally, integrating it into integrated development environments (IDEs) or database design tools could streamline the design process by providing real-time feedback and recommendations to developers during the design phase.

## 9 DATA AVAILABILITY

Research artifacts and experimental data are available at
https://www.doi.org/10.6084/m9.figshare.24233629

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments. This work was completed utilizing the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative. This work was supported in part by awards CCF-1755890, CCF-1618132, CCF-2139845, and CCF-2124116 from the National Science Foundation.

## REFERENCES


[1] 2024. Design Tradeoff Sculptor Experimental Data. https://www.doi.org/10.6084/m9.figshare.24233629

[2] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya. 2009. Archeopterix: An extendable tool for architecture optimization of AADL models. In *Proceedings of MOMPES*. 61–71.

[3] Apache. 2023. FreeMarker. https://freemarker.apache.org/.

[4] Sasan Azizian, Elham Rastegari, and Hamid Bagheri. 2024. Leveraging Machine Learning for Optimal Object-Relational Database Mapping in Software Systems. In *Proceedings of the 1st ACM International Conference on AI-Powered Software* (Porto de Galinhas, Brazil) *(AIware 2024)*. Association for Computing Machinery, New York, NY, USA, 94–102. https://doi.org/10.1145/3664646.3664769

[5] Hamid Bagheri, Kevin J. Sullivan, and Sang H. Son. 2012. Spacemaker: Practical Formal Synthesis of Tradeoff Spaces for Object-Relational Mapping.. In *Proceedings of the 24th International Conference on Software Engineering & Knowledge Engineering, Hotel Sofitel, Redwood City, San Francisco Bay, USA*. 688–693.

[6] Hamid Bagheri, Chong Tang, and Kevin J. Sullivan. 2014. TradeMaker: automated dynamic analysis of synthesized tradespaces. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*. 106–116. https://doi.org/10.1145/2568225.2568291

[7] Hamid Bagheri, Chong Tang, and Kevin J. Sullivan. 2017. Automated Synthesis and Dynamic Analysis of Tradeoff Spaces for Object-Relational Mapping. *IEEE Trans. Software Eng.* 43, 2 (2017), 145–163. https://doi.org/10.1109/TSE.2016.2587646

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[9] Daniel Le Berre and Stéphanie Roussel. 2014. Sat4j 2.3.2: on the fly solver configuration System Description. *J. Satisf. Boolean Model. Comput.* 8, 3/4 (2014), 197–202. https://doi.org/10.3233/sat190098

[10] Kush Bhatia, Avanika Narayan, Christopher De Sa, and Christopher Ré. 2023. TART: A plug-and-play Transformer module for task-agnostic reasoning. https://doi.org/10.48550/arXiv.2306.07536 arXiv:2306.07536 [cs].

[11] E. Bondarev, M. R. V. Chaudron, and E. A. de Kock. 2007. Exploring performance trade-offs of a JPEG decoder using the DeepCompass framework. In *Proceedings of WOSP*. 153–163.

[12] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. http://arxiv.org/abs/2204.02311 arXiv:2204.02311 [cs].

[13] Derek Colley, Clare Stanier, and Md Asaduzzaman. 2020. Investigating the Effects of Object-Relational Impedance Mismatch on the Efficiency of Object-Relational Mapping Frameworks. *J. Database Manage.* 31, 4 (oct 2020), 1–23. https://doi.org/10.4018/JDM.2020100101

[14] Koha Community. 2025. Koha Library Management System. https://koha-community.org/. An open source system coordinating book loans and library operations. Accessed: 2025-02-23.

[15] DB-Engines. 2024. DB-Engines Ranking. https://db-engines.com/en/ranking. https://db-engines.com/en/ranking

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[17] DjangoORM. 2023. Django ORM website. https://docs.djangoproject.com/en/4.2/ref/databases/.

[18] Steve Ebersole, Gail Badner, Andrea Boriero, and Sanne Grinovero. 2023. Hibernate website. https://hibernate.org/orm/.

[19] Y. Feng, W. Xiong, X. Wei, P. Huang, and Z. Sun. 2020. CodeBERT: A Pre-trained Model for Programming and Natural Languages. *arXiv preprint arXiv:2002.08155* (2020).

[20] Walaa Gad, Anas Alokla, Waleed Nazih, Mustafa Aref, and Abdel-badeeh Salem. 2022. DLBT: Deep Learning-Based Transformer to Generate Pseudo-Code from Source Code. *Computers, Materials & Continua* 70, 2 (2022).

[21] Aayush Garg, Milos Ojdanic, Renzo Degiovanni, Thierry Titcheu Chekam, Mike Papadakis, and Yves Le Traon. 2022. Cerebro: Static subsuming mutant selection. *IEEE Transactions on Software Engineering* 49, 1 (2022), 24–43.

[22] Guo Haixiang, Li Yijing, Jennifer Shang, Gu Mingyun, Huang Yuanyue, and Gong Bing. 2017. Learning from class-imbalanced data: Review of methods and applications. *Expert systems with applications* 73 (2017), 220–239.

[23] Julio Noe Hernandez, Jesús Ariel Carrasco-Ochoa, and José Francisco Martínez Trinidad. 2013. An Empirical Study of Oversampling and Undersampling for Instance Selection Methods on Imbalance Datasets. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications - 18th Iberoamerican Congress, CIARP 2013, Havana, Cuba, November 20-23, 2013, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 8258)*, José Ruiz-Shulcloper and Gabriella Sanniti di Baja (Eds.). Springer, 262–269. https://doi.org/10.1007/978-3-642-41822-8_33

[24] Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. arXiv:1801.06146 [cs.CL]

[25] Umair Inayat. 2025. Banking Management System. https://github.com/umairinayat/Banking-management-system. Open source project handling customer data, accounts, and loans. Accessed: 2025-02-23.

[26] Christopher Ireland and David Bowers. 2015. Exposing the myth: object-relational impedance mismatch is a wicked problem. In *The Seventh International Conference on Advances in Databases, Knowledge, and Data Applications*. 21–26.

[27] Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh. 2009. A Classification of Object-Relational Impedance Mismatch. In *First International Confernce on Advances in Databases, Knowledge, and Data Applications*. 36–43. https://doi.org/10.1109/DBKDA.2009.11

[28] Daniel Jackson. 2002. Alloy: A Lightweight Object Modelling Notation. *ACM Trans. Softw. Eng. Methodol.* 11, 2 (April 2002), 256–290. https://doi.org/10.1145/505145.505149

[29] D. Jackson. 2012. *Software Abstractions* (2nd ed.). MIT Press.

[30] Nan Jiang, Thibaud Lutellier, and Lin Tan. 2021. CURE: Code-Aware Neural Machine Translation for Automatic Program Repair. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1161–1173. https://doi.org/10.1109/ICSE43902.2021.00107

[31] Eunsuk Kang, Ethan K. Jackson, and Wolfram Schulte. 2010. An Approach for Effective Design Space Exploration. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems - 16th Monterey Workshop 2010, Redmond, WA, USA, March 31- April 2, 2010, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 6662)*, Radu Calinescu and Ethan K. Jackson (Eds.). Springer, 33–54. https://doi.org/10.1007/978-3-642-21292-5_3

[32] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[33] Sean Lau. 2006. *Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates*. M.S. Thesis. Elect. Comput. Eng., Univ. Waterloo, Waterloo, ON, Canada.

[34] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Bkg6RiCqY7

[35] Kyle Mahowald, Anna A Ivanova, Idan A Blank, Nancy Kanwisher, Joshua B Tenenbaum, and Evelina Fedorenko. 2024. Dissociating language and thought in large language models. *Trends in Cognitive Sciences* (2024).

[36] A. Martens, H. Koziolek, S. Becker, and R. H. Reussner. 2010. Automatically improve software models for performance, reliability and cost using genetic algorithms. In *Proceedings of the 1st Int. Conf. on Performance Engineering*. 105–116.

[37] Meta AI. 2024. *Introducing Llama 3*. Meta AI. https://ai.meta.com/blog/meta-llama-3/

[38] Microsoft. 2020. Entity Framework documentation. https://docs.microsoft.com/en-us/ef/.

[39] Tim Nelson, Salman Saghafi, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. 2013. Aluminum: principled scenario exploration through minimality. In *ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, David Notkin, Betty H. C. Cheng, and Klaus Pohl (Eds.). IEEE Computer Society, 232–241. https://doi.org/10.1109/ICSE.2013.6606569

[40] nopCommerce Team. 2025. nopCommerce. https://github.com/nopSolutions/nopCommerce. A free and open source e-commerce platform providing core online store functionality. Accessed: 2025-02-23.

[41] OpenAI. 2023. GPT-4 Technical Report. https://doi.org/10.48550/arXiv.2303.08774 arXiv:2303.08774 [cs].

[42] Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. 2014. Using Genetic Improvement and Code Transplants to Specialise a C++ Program to a Problem Class. In *Genetic Programming*. Springer Science + Business Media.

[43] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. 2019. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066* (2019).

[44] Sorawee Porncharoenwase, Tim Nelson, and Shriram Krishnamurthi. 2018. CompoSAT: Specification-Guided Coverage for Model Finding. In *Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10951)*, Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik P. de Vink (Eds.). Springer, 568–587. https://doi.org/10.1007/978-3-319-95582-7_34

[45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (Jan. 2020), 140:5485–140:5551.

[46] Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How Much Knowledge Can You Pack Into the Parameters of a Language Model?. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 5418–5426. https://doi.org/10.18653/v1/2020.emnlp-main.437

[47] Eric Schulte, Jonathan Dorn, Stephen Harding, Stephanie Forrest, and Westley Weimer. 2014. Post-compiler Software Optimization for Reducing Energy. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. ACM, New York, NY, USA.

[48] SQLAlchemy. 2023. SQLAlchemy website. https://www.sqlalchemy.org/.

[49] Statista. 2024. Ranking of the most popular database management systems worldwide. https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/. https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/

[50] Clay Stevens and Hamid Bagheri. 2020. Reducing run-time adaptation space via analysis of possible utility bounds. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1522–1534. https://doi.org/10.1145/3377811.3380365

[51] Clay Stevens and Hamid Bagheri. 2022. Combining solution reuse and bound tightening for efficient analysis of evolving systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, South Korea) *(ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 89–100. https://doi.org/10.1145/3533767.3534399

[52] Clay Stevens and Hamid Bagheri. 2022. Parasol: efficient parallel synthesis of large model spaces. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 620–632. https://doi.org/10.1145/3540250.3549157

[53] Clay Stevens and Hamid Bagheri. 2024. Scalable Relational Analysis via Relational Bound Propagation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 177, 12 pages. https://doi.org/10.1145/3597503.3639171

[54] Allison Sullivan, Darko Marinov, and Sarfraz Khurshid. 2019. Solution Enumeration Abstraction: A Modeling Idiom to Enhance a Lightweight Formal Method. In *Formal Methods and Software Engineering - 21st International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China, November 5-9, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11852)*, Yamine Aït Ameur and Shengchao Qin (Eds.). Springer, 336–352. https://doi.org/10.1007/978-3-030-32409-4_21

[55] Chong Tang, Hamid Bagheri, Sarun Paisarnsrisomsuk, and Kevin Sullivan. 2017. Towards Designing Effective Data Persistence through Tradeoff Space Analysis. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 353–355. https://doi.org/10.1109/ICSE-C.2017.106

[56] Vinny J. Thompson. 2025. mySDB: An Open Source Boy Scout Database. https://github.com/vinnyjth/mySDB.

[57] Patanamon Thongtanunam, Chanathip Pornprasit, and Chakkrit Tantithamthavorn. 2022. AutoTransform: Automated Code Transformation to Support Modern Code Review Process. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 237–248. https://doi.org/10.1145/3510003.3510067

[58] Emina Torlak and Daniel Jackson. 2007. Kodkod: A Relational Model Finder. In *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4424)*, Orna Grumberg and Michael Huth (Eds.). Springer, 632–647. https://doi.org/10.1007/978-3-540-71209-1_49

[59] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. http://arxiv.org/abs/2302.13971 arXiv:2302.13971 [cs].

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[61] Jianghao Wang, Hamid Bagheri, and Myra B. Cohen. 2018. An evolutionary approach for analyzing Alloy specifications. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE'18)*. Association for Computing Machinery, New York, NY, USA, 820–825. https://doi.org/10.1145/3238147.3240468

[62] Jianghao Wang, Clay Stevens, Brooke Kidmose, Myra B. Cohen, and Hamid Bagheri. 2024. Evolutionary Analysis of Alloy Specifications with an Adaptive Fitness Function. In *Search-Based Software Engineering*, Gunel Jahangirova and Foutse Khomh (Eds.). Springer Nature Switzerland, 1–17.

[63] Wenxuan Wang, Wenxiang Jiao, Yongchang Hao, Xing Wang, Shuming Shi, Zhaopeng Tu, and Michael R. Lyu. 2022. Understanding and Improving Sequence-to-Sequence Pretraining for Neural Machine Translation. In *Proceedings of the*

*60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 2591–2600. https://doi.org/10.18653/v1/2022.acl-long.185

[64] Yue Wang, Ronny Shin, Pengcheng Yin, Graham Neubig, Jie Liu, Shuyan Wang, Siyan Ma, Guangping Jin, Xiaoyuan Sun, and Tao Zhang. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

[65] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[66] Guolong Zheng, Hamid Bagheri, Gregg Rothermel, and Jianghao Wang. 2020. Platinum: Reusing Constraint Solutions in Bounded Analysis of Relational Logic. In *Fundamental Approaches to Software Engineering (FASE)*. Springer, 29–52. https://doi.org/10.1007/978-3-030-45234-6_2

[67] E. Zitzler and L. Thiele. 1999. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *Trans. Evol. Comp* 3, 4 (nov 1999), 257–271. https://doi.org/10.1109/4235.797969

[68] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. da Fonseca. 2003. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation* 7, 2 (2003), 117–132. https://doi.org/10.1109/TEVC.2003.810758

Received 2025-02-26; accepted 2025-04-01