Machine Learning for Constraint-based Configuration: A Survey

CHRISTIAN BÄHNISCH*, Hamburger Informatik Technologie-Center e.V., Germany
ALEXANDER FELFERNIG†, Graz University of Technology, Austria
DAMIAN GARBER‡, Graz University of Technology, Austria
ALBERT HAAG, Product Management Haag GmbH, Germany
DENIS HELIC, Graz University of Technology, Austria
LOTHAR HOTZ, Hamburger Informatik Technologie-Center e.V., Germany
VIET-MAN LE, Graz University of Technology, Austria
SEBASTIAN LUBOS, Graz University of Technology, Austria

Constraint-based configuration is a successful industrial application of symbolic Artificial Intelligence. It involves selecting a set of components, features, or services that satisfy a given set of user requirements. These requirements, specified by an individual user or a group, guide the configuration system in identifying a solution that aligns with both, user requirements and the constraints defined in the configuration knowledge base. As configuration tasks grow in size and complexity, there is a growing need to integrate machine learning (ML) for increasing algorithmic efficiency and quality of user interaction. This survey provides a comprehensive overview of approaches that combine ML with constraint-based configuration techniques. We highlight key developments including new developments related to the integration of Large Language Models (LLMs) and identify open research challenges.

JAIR Track: Constraint Programming and Machine Learning

JAIR Associate Editor: Tias Guns

JAIR Reference Format:
Christian Bähnisch, Alexander Felfernig, Damian Garber, Albert Haag, Denis Helic, Lothar Hotz, Viet-Man Le, and Sebastian Lubos. 2026. Machine Learning for Constraint-based Configuration: A Survey. *Journal of Artificial Intelligence Research* 86, Article 30 (July 2026), 24 pages. DOI: 10.1613/jair.1.21207

*Corresponding Author.
†Corresponding Author.
‡Corresponding Author.

Authors' Contact Information: Christian Bähnisch, ORCID: 0009-0007-7859-9585, christian.baehnisch@hitec-hamburg.de, Hamburger Informatik Technologie-Center e.V., Hamburg, Hamburg, Germany; Alexander Felfernig, ORCID: 0000-0003-0108-3146, alexander.felfernig@tugraz.at, Graz University of Technology, Graz, Styria, Austria; Damian Garber, ORCID: 0009-0005-0993-0911, damian.garber@tugraz.at, Graz University of Technology, Graz, Styria, Austria; Albert Haag, ORCID: 0000-0002-1388-4904, albert@product-management-haag.de, Product Management Haag GmbH, Bad Dürkheim, Rheinland-Pfalz, Germany; Denis Helic, ORCID: 0000-0003-0725-7450, dhelic@tugraz.at, Graz University of Technology, Graz, Styria, Austria; Lothar Hotz, ORCID: 0000-0001-7370-7726, lothar.hotz@hitec-hamburg.de, Hamburger Informatik Technologie-Center e.V., Hamburg, Hamburg, Germany; Viet-Man Le, ORCID: 0000-0001-5778-975X, v.m.le@tugraz.at, Graz University of Technology, Graz, Styria, Austria; Sebastian Lubos, ORCID: 0000-0002-5024-3786, sebastian.lubos@tugraz.at, Graz University of Technology, Graz, Styria, Austria.

DOI: 10.1613/jair.1.21207

1 Introduction

Constraint-based configuration (Becker and Fargier 2012; E. C. Freuder and O'Sullivan 2001; Junker 2006; Mailharro 1998; Mittal and Frayman 1989) is a successful industrial application of knowledge-based Artificial Intelligence. It can be regarded as a core technology of different business processes (ranging from sales configuration to a detailed technical configuration on the level of production) and is widely applied in various industrial domains, including railway interlocking systems (Falkner, Friedrich, Haselböck, et al. 2016), automotive manufacturing (Astesana et al. 2010), and software systems (Felfernig, Falkner, et al. 2024). The basic configuration objective is to identify a valid set of components or services that meet both, the constraints of a declarative product-domain model and the specified user (customer) requirements.

Existing configuration environments employ various knowledge representations, including constraint satisfaction problems (CSP) (Junker 2006), Boolean satisfiability problems (SAT) (Biere et al. 2021), answer set programs (ASP) (Falkner, Friedrich, Schekotihin, et al. 2018), and description logics (McGuinness and Wright 1998). This survey focuses on constraint-based configuration knowledge representations. A configuration, i.e., a solution to a configuration problem, consists of a selected set of components or services, accompanied by an explanation justifying their inclusion or exclusion (Gupta et al. 2021). As configuration knowledge bases grow in size and complexity, their development and maintenance become increasingly challenging for knowledge engineers. Likewise, the expanding range of configuration options adds complexity to the decision-making process for configurator users (Felfernig, Friedrich, et al. 2004; Haag and Riemann 2011). In order to be able to effectively support configuration processes, the *integration of machine learning* (Bertolini et al. 2021) is needed in different contexts ranging from the *acquisition of constraints* (i.e., configuration knowledge engineering), *performance optimization and learning of heuristics* specifically in interactive configuration, to the *learning of user preferences* to assure user-individual and personalized configurations.

In this article, we provide an overview of existing approaches that combine machine learning (ML) with constraint-based configuration. Building on the idea of a kind of 'bilateral AI' (Hochreiter 2022), which focuses on leveraging the synergy between knowledge-based AI and ML, we provide a comprehensive survey of ML-enhanced configuration techniques. These techniques support various tasks, including configuration *knowledge engineering* (i.e., supporting the formalization of a configuration domain), proactive *user assistance* during configurator interactions (i.e., to guide the user and provide help in the case of inconsistent requirements), and the *reconfiguration* of existing configurations to take into account new requirements (i.e., help to find the most reasonable adaptations to already existing configurations). We will take into account application scenarios of generative AI – specifically, Large Language Models (LLMs) (H. Li 2022) – which becomes an increasingly important technology in constraint-based systems development and maintenance. This survey examines the integration of ML with knowledge-based configuration techniques and offers a structured perspective that complements existing configuration-related overviews (Falkner, Felfernig, et al. 2011; Junker 2006; Popescu et al. 2022; Sabin and Weigel 1998; Sinz et al. 2007; Stumptner 1997). Compared to these overviews, the major contributions of this article are the following: (1) we fully cover the three important configuration-related tasks of knowledge engineering, interactive configuration, and reconfiguration, (2) we include a detailed discussion of the tasks of constraint acquisition, integration of LLMs, and the support of personalized user interaction, (3) and we provide a classification of the underlying ML approaches.

The remainder of this survey is structured as follows: in Section 2, we define the scope of our survey and introduce basic concepts related to the topics of knowledge-based configuration and machine learning (ML). Section 3 explores the application of ML techniques in configuration knowledge engineering. In Section 4, we discuss the role of ML in interactive configuration. Section 5 provides an overview of ML approaches for reconfiguration. Open research issues are examined in Section 6. Finally, Section 7 presents concluding remarks.

2 Basic Concepts

In this section, we introduce the foundational concepts that serve as prerequisites for the discussions in the subsequent sections. We begin by presenting core definitions relevant to the field of knowledge-based configuration. Next, we provide a structured overview of machine learning techniques that have been applied in the literature to support configuration tasks. Finally, we delineate the scope of this survey by outlining the major configuration-related tasks considered most relevant for this review.

2.1 Knowledge-Based Configuration

A *configuration knowledge base* captures the essential product configuration knowledge, specifying which components and services can, must, or must not be combined to create a valid solution (configuration). Several approaches can be used to represent this knowledge. For example, when using constraint satisfaction problems (CSPs) (Rossi, Beek, et al. 2006), a configuration task is defined by a set of finite-domain variables that describe the configurable product (e.g., *car type* and *engine* in a car configuration) and a corresponding set of constraints (e.g., a *city car* must not be equipped with a *skibag*). Based on this definition, a *configurator*, supported by a constraint solver, seeks a solution that satisfies the given user requirements (e.g., the user prefers a city car). Alternatively, configuration tasks can be represented as *Boolean satisfiability problems* (SAT) (Biere et al. 2021), where variables are of type Boolean and each domain value of a CSP variable (e.g., city, limo, combi, xdrive for car type) is represented by a corresponding Boolean variable. Another approach uses *answer set programming* (ASP) (Falkner, Friedrich, Schekotihin, et al. 2018), where reasoning is based on SAT solving; thus, ASPs are translated into SAT-based representations for solution search. A key advantage of ASP is its flexibility in knowledge representation, allowing for object-oriented modeling (Falkner, Ryabokon, et al. 2015). Finally, configuration knowledge can be represented using description logics (DL), which model the knowledge as an ontology (McGuinness and Wright 1998). Note that real-world configuration requires a dynamic process that extends the basic configuration task. Unlike pure constraint solving, the requirements typically cannot be provided by the user in a single step – they must be acquired in a *conversational process* where configuration requirements are determined within an interactive preference construction process (Falkner, Haselböck, et al. 2020; Günter and Cunis 1992). This process involves the derivation of components from the knowledge base that need to be integrated into the configuration.

To facilitate our discussion, we introduce a definition of a *configuration task* and its corresponding solution (configuration) (Felfernig, Falkner, et al. 2024). We assume that configuration tasks are represented as constraint satisfaction problems – see Definitions 1 and 2.

Definition 1. *A configuration task is defined as a constraint satisfaction problem (V, D, C), where $V = \{v_1..v_n\}$ is a set of finite-domain variables, $D = \{dom_{v1}..dom_{vn}\}$ a set of corresponding domain definitions, and $C = REQ \cup KB$ is a set of constraints. Here, $KB = \{c_1, \ldots, c_k\}$ represents a set of domain constraints, while $REQ = \{c_{k+1}, \ldots, c_l\}$ is a set of user requirements. The triple (V, D, KB) is denoted as configuration knowledge base.*

The constraints in *REQ* are often denoted as *soft constraints* (or preferences) and the constraints in *KB* are *hard constraints* specifying the underlying product domain knowledge (Junker 2006).

Example 1. *Table 1 presents a simplified example of the constraints in a car configuration knowledge base. In this example,* type *refers to the car type,* fuel *represents the average fuel consumption,* ski *indicates the presence of a skibag feature,* 4wh *denotes a 4-wheel drive, and* pd *corresponds to parking distance control (y=yes, n=no).*

Definition 2. *A configuration CONF is a set of variable assignments $\{v_1 = a(v_1), v_2 = a(v_2), .., v_k = a(v_k)\}$, where $a(v_i)$ represents the value assigned to variable v_i. CONF is consistent if $CONF \cup REQ \cup KB$ is consistent, i.e., a solution can be found. CONF is complete if every variable in V has a corresponding assignment in CONF. Finally, CONF is valid if it is both, consistent and complete.*

Table 1. Configuration task based on (Felfernig, Schubert, and Zehentner 2012): $KB = \{c_1..c_5\}$, $REQ = \{c_6, c_7\}$, and $V = \{type, fuel, ski, 4wh, pd\}$ with the domains $dom_{type} = \{city, limo, combi, xdrive\}$, $dom_{fuel} = \{4l, 6l, 10l\}$, $dom_{ski} = \{y, n\}$, $dom_{4wh} = \{y, n\}$, $dom_{pd} = \{y, n\}$.

ID	Constraint
c_1	$4wh = y \rightarrow type = xdrive$
c_2	$ski = y \rightarrow type \neq city$
c_3	$fuel = 4l \rightarrow type = city$
c_4	$(fuel = 6l \vee fuel = 4l) \rightarrow type \neq xdrive$
c_5	$type = city \rightarrow fuel \notin \{6l, 10l\}$
c_6	$4wh = n$
c_7	$ski = n$

Example 2. *An example of a valid configuration is the following:* $CONF = \{type = city, fuel = 4l, ski = n, 4wh = n, pd = y\}$.

2.2 Machine Learning In Knowledge-Based Configuration

Machine learning (ML) methods are commonly classified according to the type and availability of data used for training (Table 2 provides a related overview).

Table 2. Basic categorization of machine learning methods used in the configuration context.

ML Category	Representative Methods
Supervised Learning	Decision Trees (DT), Neural Networks (NN), Latent Factor Models (LF), In-Context Learning (ICL), Inductive Learning (IL)
Semi-Supervised Learning	Active Learning (AL)
Unsupervised Learning	Similarity-based Methods (SIM), Probabilistic Modeling (PM), Clustering (CL), Association Rule Mining (AR)
Reinforcement Learning	Model-based Reinforcement Learning (RL)
Evolutionary/Heuristic Learning	Genetic Algorithms (GA)

Supervised Learning. Relies on labeled data, where each input has a corresponding output and algorithms learn to predict outputs from inputs. Classical examples include *decision trees* (DT), for example, for optimizing solver runtime performance (O'Sullivan et al. 2004), *neural networks* (NN), and *latent factor models* (LF) such as *matrix factorization* (Koren et al. 2009), for estimating the relevance of new components in reconfiguration tasks (Felfernig, Falkner, et al. 2024). *In-context learning* (ICL) and *inductive learning* (IL) can be viewed as specialized forms of supervised learning: in-context learning enables a model to infer a pattern based on supervision at inference time, while inductive learning refers to deriving general rules from labeled training data that support prediction on unseen instances, for example, learning a configuration knowledge base from positive and negative examples (Bessiere, Coletta, Koriche, et al. 2006).

Semi-supervised Learning. Uses a combination of labeled and unlabeled data which often improves performance when labeled data is scarce. *Active learning* (AL) is a representative technique in this category that selects the most informative examples for labeling – an example is the generation of membership queries in the context of constraint acquisition (Bessiere, Coletta, Hebrard, et al. 2013). By iteratively querying the most relevant instances, AL reduces annotation costs while steadily refining the model's understanding of the underlying structure. This targeted acquisition strategy not only accelerates convergence but also helps avoid redundant or uninformative samples which makes AL particularly valuable, since manual labeling is often expensive.

Unsupervised Learning. Analyzes data without labeled outputs to discover related patterns. *Similarity-based approaches* (SIM) derive, for example, customer requirements from the preferences of similar users (Falkner, Felfernig, et al. 2011). *Probabilistic modeling* (PM) estimates, for example, the probability that a component will be of interest to the user. *Clustering* (CL), for example, groups similar problems to enable the reuse of problem-solving knowledge. *Association rule mining* (AR) is applied, for example, to learn constraints from example instances. Similarity-based approaches (SIM) include case-based reasoning (CBR) (Kolodner 1993) and collaborative filtering (CF) (Herlocker et al. 2000). The former can be viewed as a special case of the latter when similarity between configurations incorporates user choice data. For example, recommending features based on popular or co-selected configurations closely parallels collaborative filtering (Falkner, Felfernig, et al. 2011).

Reinforcement Learning. (RL) learns by interacting with an environment and receiving feedback to optimize long-term objectives. An example thereof is the reconfiguration of complex production environments to optimize their performance for upcoming production scenarios (Huang et al. 2024). In such a scenario, feedback can be received from a digital twin that represents the production environment and evaluates different proposed reconfigurations. Through iterative trial-and-error, RL adapts its strategy to maximize the cumulative reward (e.g., in terms of production efficiency).

Evolutionary and Heuristic Approaches. Related approaches such as *genetic algorithms* (GA) use optimization principles inspired by natural evolution or domain-specific heuristics. In configuration, an example application is the search for potential conflicts in configuration spaces, aiming to identify and collect conflicts before a configurator is deployed in productive use (Le et al. 2026). The resulting knowledge about so-called no-goods can then be reused, i.e., conflict resolution can rely on pre-detected conflicts without running conflict detection during each configuration session.

2.3 Scope Of This Survey

This survey focuses on research that combines machine learning (ML) with techniques relevant in constraint-based configuration. The major relevant tasks in configuration scenarios range from configuration knowledge engineering to interactive configuration and reconfiguration (see the overview in Table 3). Within *configuration knowledge engineering*, ML supports tasks such as generating knowledge bases, leveraging human input to improve model accuracy, and optimizing search heuristics by learning from the configuration space. Also, large language models (LLMs) have been applied to automate the generation of configuration knowledge bases. In the context of *interactive configuration*, ML techniques help to recommend suitable requirements, to diagnose and repair inconsistent user inputs, and to improve the overall efficiency of conflict detection and resolution. Finally, in the context of *reconfiguration*, ML supports the recommendation of configuration adaptations and the identification of new configuration features that should be used to extend existing configurations.

3 Machine Learning For Configuration Knowledge Engineering

The correctness of a configuration knowledge base – particularly its semantic consistency with real-world constraints – is crucial for a successful application in practical settings. At the same time, the development and

Table 3. Overview of configuration-related ML approaches (AL = Active Learning, AR=Association Rule Mining, DT=Decision Trees, RL=Model-based Reinforcement Learning, ICL=In Context Learning, LF=Latent Factor Models, GA=Genetic Algorithms, CL=Clustering, SIM=Similarity-based Learning, PM=Probabilistic Modeling, IL=Inductive Learning, NN=Neural Networks).

Task	Research Contribution (ML Approach)
Configuration Knowledge Engineering	***Learning Configuration Knowledge Bases*** (Acher et al. 2018) (DT), (Bessiere, Coletta, Koriche, et al. 2006) (IL), (Bessiere, Coletta, O'Sullivan, et al. 2007) (AL), (Bessiere, Coletta, Hebrard, et al. 2013) (AL), (Bessiere, Carbonnel, et al. 2023) (AL), (Gibson et al. 2008) (AR), (Shchekotykhin and Friedrich 2009) (AL), (Temple et al. 2016) (DT), (Tsouros, Berden, et al. 2023) (AL), (Ulz et al. 2017) (AL)
	Generating Configuration Knowledge Bases with LLMs (Bähnisch et al. 2025) (ICL), (Galindo et al. 2023) (ICL), (Kadioglu et al. 2024) (ICL), (Mechqrane et al. 2024) (AL), (Michailidis et al. 2024) (ICL), (Michailidis et al. 2026) (ICL),(Penco et al. 2025) (ICL), (Shi et al. 2025) (ICL), (Y. Song and Cohen 2026) (ICL), (Szeider 2025) (ICL)
	Learning to Improve Search Efficiency (Erdeniz, Felfernig, Atas, et al. 2017) (CL), (Garber et al. 2023) (SIM), (Koriche et al. 2022) (RL), (Col and Teppan 2017) (GA), (W. Song et al. 2022) (RL), (O'Sullivan et al. 2004) (DT)
Interactive Configuration	***Predicting User Preferences*** (Cöster et al. 2002) (SIM), (Erdeniz, Felfernig, Samer, et al. 2019) (LF), (Falkner, Felfernig, et al. 2011) (SIM), (Uta, Felfernig, et al. 2022) (NN)
	Predicting Conflicts and Diagnoses (Felfernig, Schubert, and Reiterer 2013) (SIM), (Le et al. 2026) (GA), (Uta, Le, et al. 2025) (NN)
	Acquiring User Preferences (Chambua et al. 2019) (LF), (X. Li et al. 2025) (NN), (Montazeralghaem et al. 2025) (PM), (Mirzadeh et al. 2005) (PM), (Y. Wang et al. 2022) (NN)
Reconfiguration	***Recommending Adaptations*** (Felfernig, Schubert, and Reiterer 2013) (SIM), (Huang et al. 2024) (RL), (Wildstrom et al. 2007) (IL)
	Recommending Extensions (Cöster et al. 2002) (SIM), (Falkner, Felfernig, et al. 2011) (SIM) , (Felfernig, Falkner, et al. 2024) (LF)

maintenance of such knowledge bases often requires substantial effort (Nordlander et al. 2007; Popescu et al. 2022). Machine learning can provide a range of support services to increase the efficiency of knowledge engineering.

3.1 Learning Configuration Knowledge Bases

Due to the increasing size and complexity of configuration knowledge bases, related development and maintenance tasks become more complex and trigger a higher demand for constraint learning/acquisition support (De Raedt et al. 2018; E. Freuder and O'Sullivan 2014; E. C. Freuder 2024, 1997; Kogler et al. 2024; Prestwich et al. 2021; Rossi and Sperduti 2004). The learning of a configuration knowledge base can be regarded as a form of inductive learning (of a constraint theory), where the knowledge base is generated from a given set of examples (Muggleton 1991). Learning is performed on the basis of test cases (examples). In addition to test cases, feedback can also be collected directly from experts and/or user communities in the context of active and community-based learning scenarios (Ulz et al. 2017).

Passive Learning. If a system solely learns a set of constraints (the knowledge base) on the basis of a set of pre-defined positive and/or negative examples, this can be regarded as a form of *passive learning* where examples stem, for example, from previously completed configurations or have been directly provided by users (before the constraint learning process is started). Consequently, passive learning infers constraints without interacting with the user (Bessiere, Coletta, Koriche, et al. 2006). In addition to examples, a so-called *learning bias B* is created which represents the space of possible constraints that can be included into the learned knowledge base. If the set of provided test cases is small or lacks diversity, this could result in situations where different learned constraint sets support the provided examples (Bessiere, Carbonnel, et al. 2023; Shchekotykhin and Friedrich 2009). A weakness of passive learning based approaches is that users need to provide the examples.

Table 4 depicts a simplified example of the basic idea of *constraint acquisition.* A constraint bias (B) is the enumeration of all possible constraints that should be used for constraint acquisition, i.e., the construction of the learned constraint network C. In our example, this is the set of binary constraints ($\rightarrow$) that can be defined over the variables $V = \{type, fuel\}$ with = as allowed relation. For simplicity, we assume $dom_{type} = \{city\}$ and $dom_{fuel} = \{4l, 6l, 10l\}$. Furthermore, we restrict the usage of $type$ to the *lhs* and the usage of $fuel$ to the *rhs*. B has been initialized with the constraints $B = \{c_1, c_2, c_3\}$. A necessary precondition for starting the learning process is that the positive and negative examples do not interfere which other, i.e., each positive example has to be consistent with all negated negative examples: $\forall e^+ \in E^+ : consistent(\{e^+\} \cup \bigwedge_{e^- \in E^-} \neg e^-\})$. In the example of Table 4, this condition is fulfilled, since $\{type = xdrive \wedge fuel = 10l\} \cup \{\neg type = xdrive \vee \neg fuel = 4l\}$ is consistent. For the positive example e^+, we can delete all constraints $c_i \in B$ where inconsistent($\{e^+\} \cup \{c_i\}$) which results in $B = B - \{c_2, c_3\}$. The reason behind is that constraints rejecting positive examples are not relevant anymore for the learning process. Furthermore, for the negative example e^-, we can add all constraints $c_i \in B$ to C where inconsistent($\{c_i\} \cup \{e^-\}$) – this results in $C = \{c_1\}$. In this context, we assume *subset minimality of negative examples* e^-, i.e., only the variable value assignments part of a conflict are part of e^- – for related details, we refer to (Bessiere, Coletta, Hebrard, et al. 2013).

Table 4. A basic constraint acquisition example. The bias B defines all potential binary constraints. The two examples $e^+ \in E^+$ and $e^- \in E^-$ are the basis for the constraint acquisition process. In the general case, more than one positive and negative example are provided.

Aspect	**Example**
bias (B)	B=$\{c_1 : type = xdrive \rightarrow fuel = 10l, c_2 : type = xdrive \rightarrow fuel = 6l, c_3 : type = xdrive \rightarrow fuel = 4l\}$
examples (E^+ and E^-)	$E^+ = \{e^+ : type = xdrive, fuel = 10l\}$ $E^- = \{e^- : type = xdrive, fuel = 4l\}$
learned constraints (C)	$e^+ \Rightarrow B = B - \{c_2, c_3\}$ $e^- \Rightarrow B = B - \{c_1\}, C = C \cup \{c_1\}$

Active Learning. In contrast to passive learning, active learning is based on the idea of system-generated examples which are then classified by the user as being solutions or non-solutions (Bessiere, Coletta, Hebrard, et al. 2013). Such examples are often denoted as *membership queries* (Angluin 1988), i.e., a user has to evaluate if a specific example can be regarded as a consistent configuration. A major advantage of active learning is that users are relieved from the task of example generation. Note that example evaluators do not necessarily have to be humans. In the configuration context, example evaluations (i.e., answering membership queries) can also be supported by legacy configurators, i.e., configurators to be substituted by new ones with a different type of

knowledge representation. In this context, the "old" configurator can be used as an external system/oracle (Liem and Panichella 2020) that can provide answers to membership queries when creating a new knowledge base.

Active learning can be regarded as machine learning focusing on a query-driven approach where users have to answer questions regarding the consistency of complete or partial examples. The constraint acquisition approach introduced by (Bessiere, Coletta, O'Sullivan, et al. 2007) generates examples (complete solutions/configurations) and then asks the user about the membership of the generated examples (solution or non-solution). The corresponding user responses help to guide the generation of new examples which should be irredundant to avoid situations where users have to answer redundant queries (Bessiere, Coletta, O'Sullivan, et al. 2007). The number of such membership queries can be exponentially large, i.e., analyzing complete configurations (queries) can become a quite effortful task.

On the basis of (Bessiere, Coletta, O'Sullivan, et al. 2007), further work has introduced the concept of partial (membership) queries which support the user in focusing on minimal conflicts, i.e., minimal factors that make an example a negative one (Bessiere, Coletta, Hebrard, et al. 2013). The approach resembles the determination of subset-minimal conflict sets on the basis of QuickXPlain (Junker 2004). To tackle the challenge of reducing the number of queries in constraint acquisition, probabilistic classification has been applied to predict model membership of candidate constraints (Tsouros, Berden, et al. 2023). Large language models (LLMs) can be applied to reduce the number of needed queries in interactive constraint acquisition – this can be achieved by enabling textual user feedback to go beyond basic yes/no answers regarding the consistency of proposed examples (Mechqrane et al. 2024).

A related line of research is the automated testing and debugging of configuration knowledge bases, where examples (in terms of positive and negative test cases) are used to identify faulty constraints in the knowledge base (Felfernig, Friedrich, et al. 2004). The underlying idea is to apply the concepts of model-based diagnosis (Reiter 1987) to identify minimal sets of constraints to be deleted from the knowledge base (to satisfy all positive examples) and to integrate extensions to the knowledge base such that all negative examples get rejected. A related approach in the context of configuration space learning is introduced in (Temple et al. 2016) where a decision tree is derived from positive and negative example video generator parameterizations (a negative example is a configuration with a related low-quality video). The decision tree is then used to infer incompatibility constraints that help to avoid settings with low-quality video outcomes. A similar approach is proposed by (Acher et al. 2018) for configuring relevant content templates for technical documents.

A constraint learning approach based on association rule mining is presented in (Gibson et al. 2008) where association rules (representing candidates for compatibility constraints and component selection criteria) are derived from a training set of already completed consistent configurations. The derived rules are evaluated by domain experts with the goal to filter out incorrect ones. Finally, (Ulz et al. 2017) propose a knowledge acquisition approach based on the concepts of human computation (Ahn 2007) where constraints are generated by integrating the answers to pre-scheduled user-queries (so-called micro-tasks). Answers to these queries describe the relationship between user requirements and components. Such answers are aggregated into generalized requirement constraints which describe in which context which items should be recommended to a user.

3.2 Generating Configuration Knowledge Bases With LLMs

Large language models (LLMs) (H. Li 2022) are a class of machine learning models trained on vast text corpora to capture statistical patterns and relationships in natural language. This training enables them to produce coherent and contextually appropriate outputs. LLMs can translate informal textual specifications into structured representations by leveraging these learned patterns. In the constraint solving domain, this capability allows LLMs to map natural language descriptions to formal variables, domains, and constraints. Consequently, their

use can be seen as a step toward the long-standing Holy Grail of computer science: bridging the gap between textual descriptions and formal representations (E. C. Freuder 1997; Tsouros, Verhaeghe, et al. 2023).

The capabilities of LLMs are increasingly applied in the context of generating knowledge bases. In this context, LLMs enable the integration of rich, natural language requirements and feedback from users which helps to significantly streamline constraint acquisition, for example, by reducing the reliance on formal specification and allowing more intuitive interactions (Bähnisch et al. 2025; Galindo et al. 2023; Kadioglu et al. 2024; Mechqrane et al. 2024; Penco et al. 2025). For knowledge generation and evolution purposes, different LLM-related workflows can be applied. One option is, for example, to include the whole process from knowledge base generation to solution search into one workflow. Alternatively, knowledge base generation and solution search could be regarded as different tasks where the generation part is taken over by an LLM and solution search is performed by a constraint solver (Michailidis et al. 2026; Michailidis et al. 2024; Y. Song and Cohen 2026; Szeider 2025).

In knowledge base generation, an LLM and the constraint solver cooperate where the LLM focuses on text generation and the integrated constraint solver provides feedback in terms of syntactic and semantic correctness of the generated knowledge base. In the case of faulty solutions, the solver feedback can be applied to initiate a self-correction process where new variants of the knowledge base are generated by the LLM (Bähnisch et al. 2025; Penco et al. 2025; Shi et al. 2025). An example of a process to combine LLM-based knowledge base generation with corresponding solver feedback is depicted in Figure 1. In this process, syntactic feedback is given on the basis of the syntactic errors returned by the constraint solver, whereas semantic feedback is given on the basis of non-supported test cases (Bähnisch et al. 2025). Such test cases can be positive examples that are not supported and negative examples that are erroneously supported by the generated knowledge base (CSP). Examples are represented by complete configurations that specify intended and unintended variable value combinations.

Textual knowledge representations allow domain experts with limited knowledge in configuration knowledge representation to more easily verify, refine, and extend knowledge bases. Domain experts can provide example configurations for the purpose of testing generated knowledge bases with integrated constraint solvers. Such examples can also guide the LLM during knowledge base generation by highlighting patterns or rules that must be captured. After generating a configuration knowledge base, correctness can be automatically checked by using constraint solvers or by evaluating generated configurations with previously validated (legacy) knowledge bases. This helps to ensure that the resulting configuration knowledge base is reliable and maintains backward compatibility with existing configurations.

The example LLM-based generation process[1] in Figure 1 shows how two components, an LLM and a Python interpreter with a constraint solver, act together. The former generates the source code for solving a given automotive configuration problem (as specified in Table 1), the latter checks the code for both syntactic and semantic correctness (with regard to the defined test cases). This process illustrates how integrating generative reasoning (LLM-based code synthesis) with symbolic verification (solver-based feedback) can systematically improve knowledge base correctness. This also demonstrates how potential hallucinations or, more accurately, confabulations (Smith et al. 2023) in LLMs can be mitigated by employing a constraint solver as an exact verifier of generated artifacts.

3.3 Learning To Improve Search Efficiency

The efficiency of solution search is largely determined by the availability of effective solver search heuristics (Koriche et al. 2022). These heuristics can either be defined statically, independent of the specific configuration task, or dynamically, adapting to the current context of the configurator during search. Common search heuristics in constraint solving include variable and variable value ordering heuristics. ML approaches for learning search

[1]Find the complete LLM prompt and generated code here: https://doi.org/10.5281/zenodo.17750641.

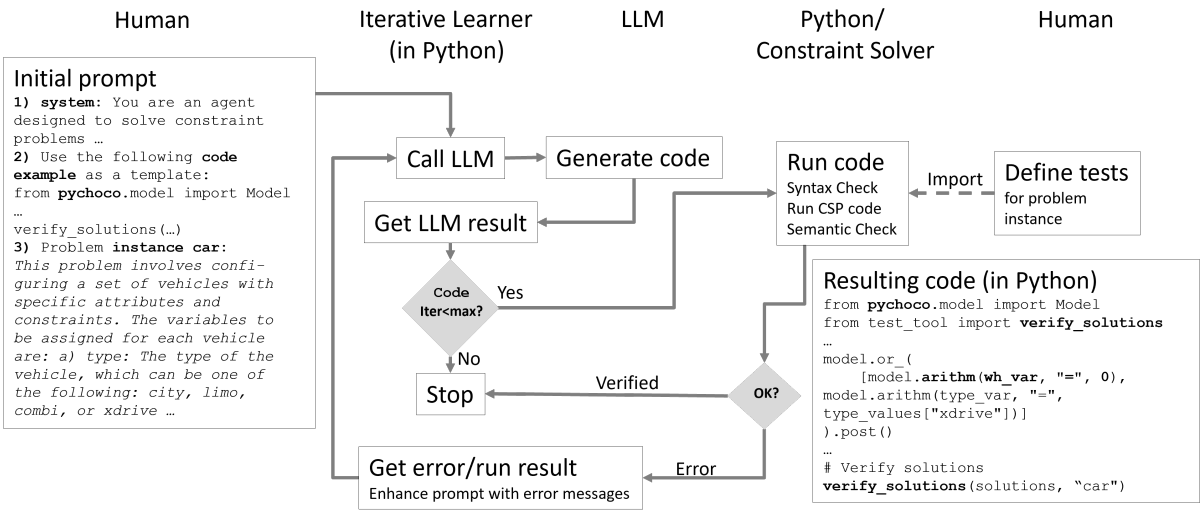

Fig. 1. Process of an LLM-based generation of a configuration knowledge base (Bähnisch et al. 2025). The *initial prompt* (lhs) (1) defines the role of the LLM, (2) includes an example source code that integrates the constraint solver and a semantic verification check, and (3) includes a description of the configuration task. The LLM generates code, which is then executed with a Python interpreter. This interpreter checks the syntax, calls the constraint solver for computing solutions, and by calling the verification method checks the generated solutions with a predefined test function (rhs). Error messages are appended to the LLM prompt triggering the next iteration of the knowledge base generation process. If the LLM generates no source code or a maximum number of iterations is reached, the knowledge base generation process ends.

heuristics exploit the concepts of *configuration space learning* (Pereira et al. 2021) where test data is synthesized on the basis of sampling strategies – the resulting test data is then used for training search heuristics.

Learning Nogood Representations. In most configuration settings, the underlying CSPs are solved many times (O'Sullivan et al. 2004). User requirements are typically represented as a set of variable value assignments which are forwarded to a constraint solver for the purpose of checking the consistency of the requirements with the underlying knowledge base, i.e., whether the requirements (represented by unary constraints) can be extended to a complete solution. The solution space of a CSP, i.e., configuration problem, often has some immanent structure with some form of interchangeability among variable values. An approach to enhance constraint satisfaction problem (CSP) solving by incorporating decision trees learned from example configurations (positive and negative examples) is proposed in (O'Sullivan et al. 2004). The core idea is to learn patterns from valid and invalid assignments to be able to detect infeasible requirements early even without the need of a constraint solver. The following example decision tree (on top of our car configuration knowledge base) could have been learned on the basis of a set of positive and negative configuration examples derived from our example knowledge base (see Table 1). For the integration of decision tree learning into constraint programming see also (Bonfietti et al. 2015).

Decision trees significantly improve CSP search performance when learned patterns can confidently predict consistent variable assignments early. During CSP solving, the decision tree can *guide variable assignments* and prune invalid options. For instance, if the user defines $type = city$, the tree immediately suggests $fuel = 4l$ thus preventing exploration of inconsistent choices (O'Sullivan et al. 2004). Decision trees are most helpful in CSP search when repeatable relationships let them quickly eliminate unlikely options.

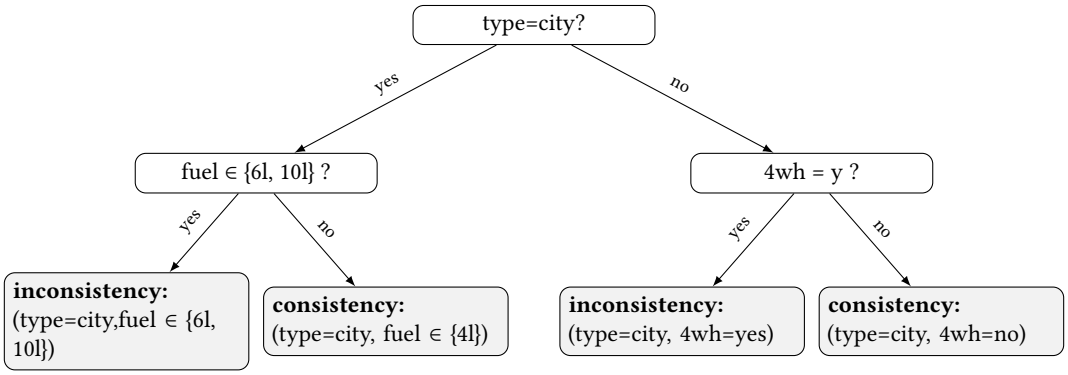

Fig. 2. Decision tree for validating combinations of *type*, *fuel*, and *4wh*.

Learning Variable and Value Orderings. Whereas decision trees follow a model-based approach (a machine learning model is built from corresponding example configurations), memory-based machine learning approaches follow the idea of identifying, for example, nearest neighbors (NNs) which are then used to infer the relevant settings for the current case (Garber et al. 2023). An example of learning search heuristics is the application of similarity-based machine learning for identifying variable value orderings. Table 5 sketches the idea of memory-based learning of variable value orderings. For generating configuration $conf_1$, the variable value ordering used by the solver was: type(city,limo,combi,xdrive), fuel(4l,6l,10l), ski(yes,no), 4wh(no,yes), and pd(yes,no). For the variable *type*, this means that the solver would try value assignments following the sequence $city \rightarrow limo \rightarrow combi \rightarrow xdrive$. In the *current* configuration session, the user has already specified preferences for the variables car *type* and *fuel* consumption. Based on this, configurations $conf_1$ and $conf_3$ are identified as the nearest neighbors in terms of preference similarity. Before initiating the search for the remaining variable value assignments, the variable orderings used for configuring nearest neighbor $conf_1$ should be reused, as they have demonstrated the best runtime performance (compared to the variable orderings of the alternative NN $conf_3$).

Table 5. Example of configuration session logs. In this example, *current* refers to the configuration session of the active user. In the already completed configuration sessions $conf_1$ – $conf_3$, varying variable value orderings have been applied. Boldface values indicate variable values that are part of the final configuration. In the current session, the user has already specified preferences for the variables *type* and *fuel*. The nearest neighbors (NN) of the current user are $conf_1$ and $conf_3$.

configuration	**type**	**fuel**	**ski**	**4wh**	**pd**	runtime (**msec**)
$conf_1$ (NN)	**c**, l, co, x	**4l**, 6l, 10l	y, **n**	**n**, y	**y**, n	9
$conf_2$	c, **l**, co, x	4l, **6l**, 10l	**y**, n	**n**, y	**y**, n	14
$conf_3$ (NN)	x, co, l, **c**	10l, 6l, **4l**	**n**, y	y, **n**	**y**, n	27
current	**c**	**4l**	n, y	y, n	y, n	–

The memory-based (similarity-based) approach exemplified in Table 5 is based on the idea of generating example configurations for a specific (pre-defined) knowledge base and using those examples to identify efficient variable value orderings (best heuristics identification). This idea can be extended in such a way that the training set is represented by different CSP instances (same CSP but different initial variable value assignments) which are then split into different clusters. Such a clustering approach is introduced in (Erdeniz, Felfernig, Atas, et al. 2017), where for each cluster, variable orderings are optimized individually. For a new (yet unsolved) CSP instance, the

most similar cluster is identified and the variable orderings optimized for CSPs in this cluster are applied to the new CSP.

While the aforementioned approaches rely on similarity and clustering to reuse effective heuristics, alternative methods focus on directly learning search strategies through adaptive optimization techniques such as reinforcement learning (W. Song et al. 2022). Beyond variable ordering, the authors emphasize the potentials of the proposed approach in terms of learning other control policies such as value ordering and propagator selection. An approach to learn search heuristics (variable and value orderings) on the basis of genetic algorithms is presented in (Col and Teppan 2017). The idea is to apply genetic algorithms to individual problem settings where individuals (the members of population) are represented on the basis of the defined variable and value orderings. An optimization function evaluates the performance of the individuals in solving the underlying constraint satisfaction problem. The problem setting defined in (Koriche et al. 2022) is similar – the overall objective is to identify the optimal variable ordering strategy based on a specific CSP definition. The heuristic optimization task is based on the representation of a multi-armed bandit problem, where each heuristic is considered an "arm" and trying it on a CSP provides a performance-based reward. The algorithm repeatedly evaluates heuristics, compares their rewards, and progressively focuses on the most promising ones using an adaptive successive-halving strategy. This is a form of reinforcement learning because the system learns which heuristic to select through interaction and observed rewards, without any labeled data or predefined correct answers.

4 Interactive Configuration With Machine Learning

After having developed a configurator application including its user interface, reasoning engine, and underlying knowledge base, it can be deployed for productive use. In this context, effective user support depends on the availability of personalization services (Ardissono et al. 2003). Such services help users in finding relevant variable values (e.g., a user does not have the background knowledge to do this) and provide support in situations where the configurator is not able to find a solution for the given user requirements.

4.1 Predicting User Preferences

As configuration knowledge bases grow in size and complexity due to an increasing number of parameters and related constraints, there is a need for techniques that proactively assist users in their preference-building process (Falkner, Felfernig, et al. 2011). Due to limited product domain knowledge and also time constraints, users often require personalized support to efficiently select the right parameter settings. ML can play a key role by predicting relevant parameter choices tailored to the user's needs. Specifically, ML algorithms can be used to predict individual variable settings based on historical data.

Recommending Variable Values. Similarity-based machine learning approaches (Cöster et al. 2002; Ekstrand et al. 2011) can leverage a dataset of consistent configurations to identify preferences of similar users. Such nearest neighbors (NN) can serve as predictors for the preferences of the current user (Falkner, Felfernig, et al. 2011) — see the example in Table 6.

More advanced approaches integrate model-based techniques, such neural networks (Uta, Felfernig, et al. 2022) or matrix factorization (Erdeniz, Felfernig, Samer, et al. 2019). The approach introduced in (Uta, Felfernig, et al. 2022) involves *feed-forward neural networks* (FFNNs), which leverage past configuration sessions and their corresponding recommendations to train the network for future value predictions. Typically, an FFNN consists of several hidden layers following the input layer, with the output layer structured as a multi-label, multi-class classification task. This output computes probability distributions over the values of each categorical variable, where each probability indicates the likelihood that a specific variable value will be relevant to the user.

Table 6. Similarity-based predictions for the current user who already selected a limousine and a skibag. Based on the identified NN, the recommender predicts **fuel = 6l** and **4wh = n**.

configuration	**type**	**fuel**	**ski**	**4wh**	**comment**
$conf_1$	city	4l	n	n	*no match*
$conf_2$ (NN)	limousine	6l	y	n	*NN: match with type and ski*
$conf_3$	combi	10l	y	n	*match with ski*
$conf_4$	xdrive	10l	y	y	*match with ski*
$conf_5$	xdrive	4l	n	y	*no match*
$conf_6$	limousine	6l	n	n	*match with type*
current	**limousine**	**?**	**y**	**?**	*customer selections*
CF prediction	limousine	**6l**	y	**n**	*predicted from NN*

Recommending Variable Value Orderings. It is important to note that ML-based variable value predictions are not necessarily consistent with *KB* and *REQ* (Falkner, Felfernig, et al. 2011). Recommendations have to be checked for consistency and alternatives have to be determined in the case of inconsistencies. As an alternative to predicting specific variable values, ML models can be designed to recommend variable value orderings which help to guarantee consistent (and personalized) solutions of relevance for the user. For example, similarity-based approaches (Garber et al. 2023) or matrix factorization (Erdeniz, Felfernig, Samer, et al. 2019) can be employed to recommend variable value orderings. Similar to the example in Table 6, nearest neighbors (NN) are identified based on the similarity between the current user's requirements and previously completed configurations by similar users. The variable value ordering used for completing the configurations of NNs are then recommended for completing the current user's configuration.

4.2 Predicting Conflicts and Diagnoses

In interactive configuration scenarios, user (customer) requirements (*REQ*) can become inconsistent with the underlying set of domain constraints (*KB*) and users need support to get out of the *no solution could be found* situation. Two major related concepts can provide help in terms of explaining the sources of an inconsistency. First, *minimal conflict sets* $CS \subseteq REQ$ are minimal sets of user requirements that induce an inconsistency with the constraints in *KB*, i.e., inconsistent($CS \cup KB$). Second, *minimal diagnoses* (i.e., conflict resolutions) are minimal sets $\Delta \subseteq REQ$ that have to be omitted or adapted in order to be able to restore consistency, i.e., consistent($REQ - \Delta \cup KB$). QuickXPlain (Junker 2004) is an often-applied algorithm for the identification of minimal conflict sets in *REQ*. It is based on the idea of divide-and-conquer: if a constraint set $C = \{c_1..c_k, c_{k+1}..c_n\}$ is *inconsistent* (e.g., $k = \frac{|\{c_1..c_n\}|}{2}$) and $\{c_1..c_k\}$ is inconsistent, then conflict search can focus on this subset and – with one consistency check – we can omit half of the constraints in C (in our case, $C = REQ$). Minimal diagnoses, i.e., conflict resolutions, can be computed, for example, on the basis of a hitting set directed acyclic graph (HSDAG) (Reiter 1987).

Example-based Personalized Diagnosis. To predict diagnoses *of relevance* for a user, various machine learning approaches can be applied (Felfernig, Schubert, and Reiterer 2013; Uta, Le, et al. 2025). The machine learning approach introduced in (Felfernig, Schubert, and Reiterer 2013) is similarity-based – it is based on a *dataset of completed and consistent configurations* where diagnosis search is personalized using a best-first HSDAG search with node expansion decisions based on metrics that estimate the similarity between a reconfiguration candidate (associated with a diagnosis) and the original user requirements (Felfernig, Schubert, and Reiterer 2013). A similarity-based ML approach in the context of direct diagnoses with FastDiag (Felfernig, Schubert,

and Zehentner 2012) is included in (Uta, Le, et al. 2025). A set of known inconsistent configurations from previous configuration sessions including the diagnoses chosen by the corresponding users is exploited to identify potential diagnoses in new configuration sessions. The idea is to identify configuration sessions in the dataset with originally inconsistent variable value selections that are similar compared to the variable value selections in the current configuration session. In the example of Table 7, the (inconsistent) configuration $conf_2$ is identified as nearest neighbor of the current (inconsistent) configuration. The diagnosis chosen in $conf_2$ can be regarded as a potential diagnosis candidate in the current configuration session. For further related details, we refer to (Uta, Le, et al. 2025).

Table 7. Dataset containing originally inconsistent configurations $conf_\alpha$ along with the corresponding diagnoses $\Delta_{\alpha\beta}$ selected by the user. For instance, to resolve the inconsistency in $conf_2$, the user chose Δ_{21} (chosen diagnoses are indicated in bold).

configuration	**type**	**fuel**	**ski**	**4wh**	**chosen diagnosis**
$conf_1$	**city**	4l	**y**	n	$\mathbf{\Delta_{11}}$:{ski=y}, Δ_{12}:{type=city}
$conf_2$ (NN)	**city**	**10l**	n	n	$\mathbf{\Delta_{21}}$:{fuel=10l}, Δ_{22}:{type=city}
$conf_3$	**xdrive**	**6l**	n	n	Δ_{31}:{type=xdrive}, $\mathbf{\Delta_{32}}$:{fuel=6l}
current	**city**	**10l**	**n**	**n**	$\mathbf{\Delta_{21}}$: {**fuel = 10l**}

Personalized Diagnosis with Model-based Machine Learning. As proposed in (Uta, Le, et al. 2025), model-based ML can be used as an alternative to similarity-based ML. Inconsistent users requirements can be ranked by a neural network (the model) in such a way that user requirements with a high probability of being accepted as diagnosis candidates are ranked correspondingly (see the sketch in Figure 3). Training data includes a set of inconsistent configurations accompanied with their corresponding diagnoses (similar to Table 7). As the goal of the neural network is to reorder requirements, several loss functions are suitable for such a ranking task. After training the model, the neural network takes an inconsistent configuration as input and outputs a relevance ranking of individual customer requirements, indicating their relevance in terms of their involvement in a diagnosis. In our example shown in Figure 3, the input for the neural network are the user requirements $\{type = city, ski = n, fuel = 10\}$. The neural network ranks those inconsistent requirements with regard to their estimated probability of being part of a diagnosis: $\{fuel(0.85), type(0.65), ski(0.40)\}$ – the higher the value, the higher the probability for the corresponding variable to be part of a chosen diagnosis. Note that such neural network based (i.e., model-based) representations have a better runtime performance compared to nearest neighbor based (i.e., memory-based) approaches which becomes relevant with an increasing dataset size.

Conflict detection and diagnosis are essential algorithmic techniques for proactively assisting users in addressing inconsistencies during configuration sessions (Felfernig, Schubert, and Zehentner 2012; Junker 2004). As discussed in the previous examples, ML-based approaches can help to efficiently identify diagnoses by calculating a preference ordering of user requirements which is used as input of the diagnosis algorithm. The goal is to reorder the requirements in such a way that the new ranking better clusters diagnosis-relevant elements. Note that in this context machine learning can focus on *different optimization functions*: for predicting diagnosis inclusion of specific user requirements, an optimization function could focus on prediction quality, i.e., $\#correct\ predictions/\#predictions$. Alternatively, when diagnosis runtime performance has the highest importance, the corresponding optimization function for model learning will focus on runtime performance. In typical configuration settings, machine learning models are based on optimization functions that represent tradeoffs between diagnosis prediction quality and performance (Uta, Le, et al. 2025).

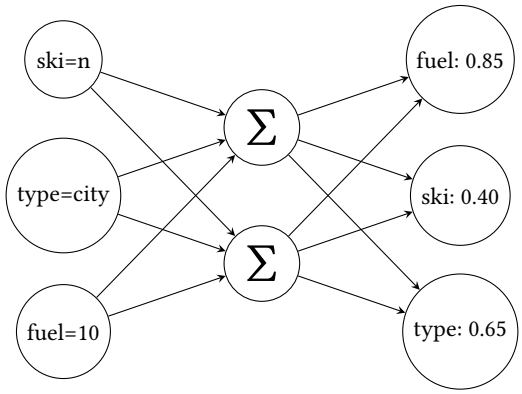

Fig. 3. Neural network used to determine an ordering of user inputs (i.e., the set of requirements) for FastDiag (Uta, Le, et al. 2025). Such rankings help to identify those diagnosis elements most relevant to the user and to arrange them accordingly for the input sequence of the diagnosis algorithm.

Lazy Conflict Detection with Genetic Algorithms. An alternative to on-demand conflict detection and diagnosis is 'offline' processing (Le et al. 2026) where potential conflicts are identified in advance of the actual use of the configuration system. A related example is presented in Table 8, where a *genetic algorithm* is employed to generate inconsistent constraint sets. The underlying approach leverages a genetic algorithm to learn how to generate inconsistent configurations. Once these configurations are identified, the minimal conflict sets are determined using QuickXPlain (Junker 2004). In the subsequent steps, the genetic algorithm takes into account the identified conflicts by avoiding these combinations in future populations. One major advantage of this approach is that conflict sets can be reused during ongoing configuration sessions which eliminates the need of activating conflict detection algorithms. This can lead to significant runtime improvements (and reduced energy consumption) compared to standard conflict detection and diagnosis methods (Le et al. 2026).

Acquiring User Preferences. The growing complexity and size of configurable products imposes significant challenges on the design of effective user interfaces (Niederer et al. 2022). While basic machine learning approaches have already been applied to assist users, for example, in terms of recommending relevant configuration variables which should be specified by the user (Mirzadeh et al. 2005), recent advancements in deep learning and large language models (LLMs) offer promising ways to enhance preference elicitation. For example, an approach that uses probabilistic matrix factorization to predict user preferences by analyzing sentiments extracted from product reviews is presented in (Chambua et al. 2019). Moreover, LLMs add a conversational, reasoning-driven layer to the configuration process, which significantly improves both, its efficiency and accuracy (Montazeralghaem et al. 2025). In the context of requirement acquisition, LLMs can function as intelligent assistants which are capable of interpreting vague or naturally expressed user needs. For instance, when a user says, "I need a car with better fuel efficiency," an LLM can extract formal parameters such as *fuel=4l*.

Recent developments in LLM-based interaction models, often augmented with retrieval-based methods, underscore the increasingly import role of conversational AI in configuration tasks (X. Li et al. 2025; Y. Wang et al. 2022). Simplifying the identification and specification of user preferences is critical for interactive configuration (Kogler et al. 2024). With their advanced capabilities in natural language understanding and generation, LLMs can play a key role in facilitating natural language-based extraction of user preferences and translating them into corresponding constraint-based representations (Kogler et al. 2024).

Table 8. One iteration of a genetic algorithm that generates inconsistent configurations. The table shows the initial population $P = \{conf_1..conf_4\}$, resulting crossovers C_α (e.g., C_1 and C_2 are the result of a crossover between $conf_1$ and $conf_2$), mutations M_β (e.g., M_1 is the result of mutating C_1), and QuickXplain (Junker 2004) generated minimal conflicts CS_γ.

phase	configuration	status	explanation
1. initial population P (type, fuel, ski, 4wh)			
$conf_1$	(city, 10l, no, no)	inconsistent	*city cannot have 10l fuel or skibag*
$conf_2$	(xdrive, 6l, no, yes)	inconsistent	*xdrive cannot have 6l fuel*
$conf_3$	(combi, 4l, no, no)	inconsistent	*4l fuel only allowed for city*
$conf_4$	(limousine, 10l, yes, no)	consistent	*valid configuration*
2. crossovers C_i **(e.g.,** $conf_1 \times conf_2$**)**			
C_1	(city, 6l, no, yes)	inconsistent	*city cannot have 6l fuel or 4wh*
C_2	(xdrive, 10l, yes, no)	inconsistent	*xdrive must have 4wh = yes*
...			
3. mutations M_i			
M_1	(city, 6l, no, no)	inconsistent	*city cannot have 6l fuel*
M_2	(xdrive, 10l, yes, yes)	consistent	*valid configuration*
...			
4. minimal conflicts CS_i			
CS_1	{type = city, fuel = 6l}	conflict	*minimal conflict*
CS_2	{type = xdrive, 4wh = no}	conflict	*minimal conflict*
...			

5 Reconfiguring With Machine Learning

Informally, reconfiguration (Männistö, Soininen, et al. 1999) refers to the modification of variable value settings in an existing configuration. For instance, a railway interlocking system may need to be reconfigured to accommodate an expanded railway infrastructure, or a company's network configuration may require adjustments to increase bandwidth to meet new demands. In the following, we assume a scenario where a user has already created an initial car configuration and now wishes to modify it. The formal definitions of a reconfiguration task and a reconfiguration are provided in Definitions 3–5.

Definition 3. *A* reconfiguration task *can be defined as a constraint satisfaction problem* $(CONF, V, D, KB, REQ_{new})$ *where* $CONF$ *is the current configuration,* (V, D, KB) *represents the configuration knowledge base, and* REQ_{new} *represents reconfiguration requirements (formulated as variable value assignments).*

Example 3. *An example of a reconfiguration task* $(CONF, V, D, KB, REQ_{new})$ *is the following:* V*,* D*, and* KB *can be taken from the configuration task definition in Table 1. Furthermore,* $CONF = \{type = city, fuel = 4l, ski = n, 4wh = n, pd = y\}$ *is a valid configuration and* $REQ_{new} = \{ski = y\}$*.*

Definition 4. *A* reconfiguration diagnosis *for a given reconfiguration task* $(CONF, V, D, KB, REQ_{new})$ *is a set* Δ *with consistent*$((CONF - \Delta) \cup KB \cup REQ_{new})$*.* Δ *is minimal if* $\neg\exists\Delta' : \Delta' \subset \Delta$ *and* Δ' *is a reconfiguration diagnosis.*

Example 4. *In our example, we are able to identify three minimal conflict sets which are* $CS_1 = \{type = city\}$*,* $CS_2 = \{fuel = 4l\}$*, and* $CS_3 = \{ski = n\}$*. This results in exactly one reconfiguration diagnosis* $\Delta = \{type = city, fuel = 4l, ski = n\}$*.*

Definition 5. *A reconfiguration RCONF for a given reconfiguration task* $(CONF, V, D, KB, REQ_{new})$ *and a corresponding reconfiguration diagnosis* Δ*, is a solution* (*configuration*) *for* $((CONF - \Delta) \cup KB \cup REQ_{new})$.

Example 5. *With the reconfiguration diagnosis* $\Delta = \{type = city, fuel = 4l, ski = n\}$*, a solver can determine five possible reconfigurations for* $(CONF - \Delta) \cup KB \cup REQ_{new}$ *– these reconfigurations are depicted in Table 9.*

Recommending Adaptations to Existing Configurations. Different basic machine learning approaches that support users in reconfiguration settings are introduced in (Felfernig, Schubert, and Reiterer 2013). A fundamental approach is to use memory-based recommendation to determine the similarity between the original user requirements and the candidate reconfigurations and present the most similar reconfiguration to the user. Assuming the existence of attribute-level similarity metrics (Felfernig, Schubert, and Reiterer 2013) that return (per configuration attribute) a similarity between 0..1, we can assume that reconfiguration $rconf_1 = \{type = limo, fuel = 6l, ski = y, 4wh = y, pd = y\}$ (see Table 9) since a limousine-type car can be assumed to be similar to a city-type car (compared to the other alternatives) and $6l$ fuel consumption is next to $4l$ (see Table 9). A noteworthy aspect in our example is that even a simple change in a single requirement, such as including a skibag, can significantly affect the user's overall configuration. In our example, a practical recommendation might be to inform the user about the option to not include the skibag and explore the possibility of solving the issue using a roof rack instead. This is a kind of *open configuration* (Chen and L. L. Zhang 2015; L. Zhang et al. 2014), where certain components are not directly included into *KB* but are still offered as flexible alternatives. Machine learning based reconfiguration is not limited to interactive configuration settings. A reconfiguration scenario in the context of complex production systems is presented in (Huang et al. 2024) where reconfigurable machine tools offer high flexibility in terms of allowing the adaptation (reconfiguration) of the production parameters to adapt to demand changes. In this context, a reinforcement learning approach is proposed where a digital twin of the manufacturing system is used to provide feedback on the quality of different reconfiguration alternatives. Finally, reconfiguration tasks have also been solved in the context of online hardware reconfiguration. Since high-end server systems are partitioned into logical subsystems, there is a need for dynamically repartitioning and optimizing performance for repeatedly changing workloads. For such a scenario, (Wildstrom et al. 2007) introduce a rule inductive learning approach where context-sensitive reconfiguration actions are encoded in a set of rules.

Table 9. Different identified reconfiguration candidates for the initial configuration $CONF = \{type = city, fuel = 4l, ski = n, 4wh = n, pd = y\}$ and the new requirements $REQ_{new} = \{ski = y\}$.

reconfiguration	type	fuel	ski	4wh	pd
$rconf_1$	limo	6l	y	n	y
$rconf_2$	limo	10l	y	n	y
$rconf_3$	combi	6l	y	n	y
$rconf_4$	combi	10l	y	n	y
$rconf_5$	xdrive	10l	y	n	y

Recommending New Features. For configurable products and services, the configuration model may evolve frequently, with outdated components removed and new ones introduced (Männistö and Sulonen 1999). If new components or features are incorporated into the configuration knowledge base, recommending these additions can be interpreted as a specific instance of predicting user preferences (Cöster et al. 2002; Falkner, Felfernig, et al. 2011) (see Section 4). The following example illustrates a basic use case of matrix factorization (MF) (Koren et al. 2009) for predicting user preferences regarding newly introduced components. This approach presupposes that a user has already configured an initial version of a product, as MF is typically performed in batch mode

for performance reasons. In this scenario, we consider a company that introduces new car components to its customers (Felfernig, Le, et al. 2021). For example, when configuring cars, an additional option could be an easily installable miniature dashcam (d), as shown in Table 10.

Table 10. *One-hot encoding* (Xiang et al. 2021) of completed configurations (matrix R) for predicting the relevance of new feature d. For configurations $\{conf_3, conf_4, conf_5\}$, a matrix factorization approach estimates interest in the new feature (higher values indicate higher (potential) interest).

configuration	c	l	co	x	4	6	10	ski	4wh	pd	d
$conf_1$	1	0	0	0	1	0	0	0	0	0	0
$conf_2$	0	0	0	1	0	0	1	1	1	1	1
$conf_3$	0	1	0	0	0	1	0	0	0	0	$? \rightarrow 0.22$
$conf_4$	0	1	0	0	0	1	0	1	0	0	$? \rightarrow 0.90$
$conf_5$	0	0	0	1	0	0	1	1	1	1	$? \rightarrow 0.98$

In this example, we assume that a dashcam (d) is designed to be compatible with all company car types, with no further restrictions regarding individual car features (we assume that it has already been included in configuration $conf_2$). Using matrix factorization (MF) (Koren et al. 2009), the entries in Table 10 can be reconstructed through dimensionality reduction (Felfernig, Falkner, et al. 2024). In this process, two low-dimensional matrices, P and Q, which represent the machine learning model, are used to approximate the original matrix R (R is represented by Table 10). The approximation, $\widehat{R}$, results from the matrix multiplication of P and Q and is given by $\widehat{R} = PQ$. In this context, P captures the "hidden" relationships between users and latent (non-interpretable) ML features, while Q represents the "hidden" relationships between these latent features and corresponding configuration features. The goal of MF is to optimize the entries in P and Q such that $\widehat{R}$ closely approximates R. One beneficial outcome of this process is that it also provides predictions for the missing entries in R, marked as '?'. This allows for the personalized expansion of a configuration tailored to individual users.

6 Research Issues

From our survey of the current state-of-the-art in machine learning and constraint-based configuration, several open research issues emerge.

Configuration Knowledge Engineering. In configuration knowledge engineering, user studies are needed to better understand and evaluate the cognitive complexity of knowledge structures. This understanding (expressed, e.g., on the basis of a set of metrics) would ultimately help to improve the quality of constraint formulations, understandability, and maintainability. Related insights will also help to better understand in which way questions can be asked to knowledge engineers, for example, in the context of constraint acquisition (Bessiere, Carbonnel, et al. 2023; Shchekotykhin and Friedrich 2009). Through their capabilities of natural language understanding and generating corresponding formal representations (Bähnisch et al. 2025), LLMs also have the potential to enhance the overall quality of product line scoping (Marchezan et al. 2022), which involves determining which components and features should be included in a configuration knowledge base. A major aspect in this context is that LLMs have the potential to support scoping discussions and decisions even without the need of having a formal knowledge base available for estimating the impact of potential scoping decisions.

Explanations. In terms of supporting interactive configuration, the quality of explanations must be enhanced, particularly by providing personalized explanations for users. Through their ability of adaptive text generation, LLMs have the potential to significantly increase the quality and flexibility of explanation generation. Also, in

real-time scenarios such as computer network configuration and reconfiguration, the performance of conflict detection and diagnosis needs to be further optimized – both can be regarded as specific explanation types (Dev Gupta et al. 2021). This could be achieved through intelligent reuse strategies that eliminate the need for online conflict detection and diagnosis determination. There also exists a need to tailor explanations to the needs of users – this can be achieved by taking into account models of human behavior (Verma et al. 2024).

Large Language Models. We want to emphasize the significant potential of Large Language Models (LLMs) in enhancing the productivity of configuration-related processes. For example, by leveraging natural language specifications and feedback, LLMs can streamline the automated generation and maintenance of knowledge bases (Bähnisch et al. 2025; Mechqrane et al. 2024). In interactive configuration environments, LLMs have the potential to enhance the quality of user preference elicitation. By enabling users to specify preferences in natural language, LLMs can simultaneously ensure the consistency and completeness of these inputs at the textual level. Moreover, LLMs could optimize the configuration process by recommending logical sequences for configuration steps. A key challenge in configuration is acquiring user requirements and determining the optimal moments to gather specific information. By feeding the LLM with a context-rich knowledge base or utilizing a retrieval-augmented generation, the model can predict the next logical step in the configuration process.

Graph Neural Network Integration. Apart from LLMs, another possibility for future research is the integration of configuration with graph neural networks (GNNs) (Scarselli et al. 2009). We see a few avenues for such a potential integration. For example, GNNs represent state-of-the-art approaches for similarity-based recommendation (He et al. 2020). They operate on user-item attributed bipartite graphs and extract low-dimensional representations of both, users and items. Such low-dimensional representations can be used to perform nearest neighbors search in user or item space, hence supporting a stronger flexibility in user preference prediction (Kipf and Welling 2017). Graph neural networks could be also applied in the re-ranking of customer requirements for diagnosis search. For example, knowledge bases can be modeled as so-called knowledge graphs (Ren et al. 2022) that can be embedded into low-dimensional vector spaces using GNNs (H. Wang et al. 2021).

LLM-based Configuration and Reconfiguration. LLM-based configuration and reconfiguration can offer improvements for traditional configuration settings. In scenarios such as public tenders for construction projects, LLMs have the potential to propose and validate textual specifications and help to ensure that they comply with legal regulations. When crafting offers in response to a tender, LLMs could not only help to analyze their conformity but also suggest necessary extensions or repairs based on the legal and technical specifications. Furthermore, LLMs could help to identify gaps, such as missing or ambiguous specifications, and autonomously generate requests for clarification. While this approach is promising, challenges such as hallucination must be carefully addressed to ensure reliability. In this context, LLMs should be regarded as a supportive technology that still requires the integration with basic machine learning approaches and formal reasoning.

7 Conclusions

With the goal of fostering further synergies, we presented a survey on the integration of constraint-based configuration and machine learning. The survey is structured along the three main dimensions of machine learning supported configuration knowledge engineering, machine learning for interactive configuration, and machine learning for reconfiguration. For each dimension, we reviewed representative approaches and discussed how machine learning techniques can be effectively leveraged within configuration processes.

In particular, machine learning models have shown to support tasks such as preference prediction for configurator users, the automated generation of configuration knowledge bases from textual product descriptions, and the computation of personalized reconfigurations that adapt configurations to newly expressed user requirements. These developments illustrate how learning-based methods can support constraint-based configuration.

From our analysis, several open research issues emerge. Promising directions include: LLM-based product line scoping, i.e., the automated or semi-automated identification of relevant product features to be included in a configuration knowledge base; the personalization of explanations (taking into account a users expertise and context); and adaptive configurator interfaces that dynamically tailor interaction and decision support to individual user needs. These challenges highlight the need for tighter integration between machine learning techniques and configuration methods in future systems.

Acknowledgments

This research was funded in whole or in part by the Austrian Science Fund (FWF) cluster of excellence *Bilateral AI* and by the German Federal Ministry for Economic Affairs and Climate Action within the project *IntelMOD* (intelligent modernization platform based on functional cost splitting).

References

M. Acher, P. Temple, J. Jézéquel, J. Galindo, J. Martinez, and T. Ziadi. 2018. "VaryLATEX: Learning Paper Variants That Meet Constraints." In: *12th International Workshop on Variability Modelling of Software-Intensive Systems*. ACM, Madrid, Spain, 83–88. doi:10.1145/3168365.3168372.

L. von Ahn. 2007. "Human Computation." In: *4th International Conference on Knowledge Capture* (K-CAP '07). ACM, Whistler, BC, Canada, 5–6. doi:10.1145/1298406.1298408.

D. Angluin. 1988. "Queries and Concept Learning." *Mach. Learn.*, 2, 4, 319–342. doi:10.1023/A:1022821128753.

L. Ardissono, A. Goy, M. Holland, G. Petrone, and R. Schäfer. 2003. "Customising the Interaction with Configuration Systems." In: *User Modeling 2003*. Ed. by P. Brusilovsky, A. Corbett, and F. de Rosis. Springer, Berlin, Heidelberg, 283–287. ISBN: 978-3-540-44963-8.

J.-M. Astesana, L. Cosserat, and H. Fargier. 2010. "Constraint-based Vehicle Configuration: A Case Study." In: *22nd IEEE Intl. Conference on Tools with Artificial Intelligence* (ICTAI '10). IEEE Computer Society, USA, 68–75. ISBN: 9780769542638. doi:10.1109/ICTAI.2010.19.

C. Bähnisch, L. Hotz, A. Felfernig, and S. Lubos. 2025. "Test-driven Generation of Constraint Satisfaction Problems Using Large Language Models." English. In: *27th International Workshop on Configuration (ConfWS 2025)*. Bologna. https://confws.github.io.

C. Becker and H. Fargier. 2012. "Maintaining alternative values in constraint-based configuration." In: *23rd International Joint Conference on Artificial Intelligence*. Montpellier, France, 454–460.

M. Bertolini, D. Mezzogori, M. Neroni, and F. Zammori. 2021. "Machine Learning for industrial applications: A comprehensive literature review." *Expert Systems with Applications*, 175, 114820. doi:https://doi.org/10.1016/j.eswa.2021.114820.

C. Bessiere, C. Carbonnel, et al.. 2023. "Learning constraints through partial queries." *Artif. Intell.*, 319, 103896. doi:10.1016/J.ARTINT.2023.103896.

C. Bessiere, R. Coletta, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C.-G. Quimper, and T. Walsh. 2013. "Constraint acquisition via partial queries." In: *23rd International Joint Conference on Artificial Intelligence* (IJCAI '13). AAAI Press, Beijing, China, 475–481. ISBN: 9781577356332.

C. Bessiere, R. Coletta, F. Koriche, and B. O'Sullivan. 2006. "Acquiring Constraint Networks Using a SAT-based Version Space Algorithm." In: *21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference*. AAAI Press, 1565–1568. http://www.aaai.org/Library/AAAI/2006/aaai06-251.php.

C. Bessiere, R. Coletta, B. O'Sullivan, and M. Paulin. 2007. "Query-Driven Constraint Acquisition." In: *20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. Ed. by M. M. Veloso, 50–55. http://ijcai.org/Proceedings/07/Papers/006.pdf.

A. Biere, M. Heule, H. V. Maaren, and T. Walsh. 2021. *Handbook of Satisfiability*. IOS Press, USA. ISBN: 978-1-64368-160-3.

A. Bonfietti, M. Lombardi, and M. Milano. 2015. "Embedding Decision Trees and Random Forests in Constraint Programming." In: *Integration of AI and OR Techniques in Constraint Programming*. Ed. by L. Michel. Springer, 74–90.

J. Chambua, Z. Niu, and Y. Zhu. 2019. "User preferences prediction approach based on embedded deep summaries." *Expert Systems with Applications*, 132, 87–98. doi:https://doi.org/10.1016/j.eswa.2019.04.047.

X. Chen and L. L. Zhang. 2015. "Simplified definition of open configuration." In: *International Conference on Industrial Engineering and Operations Management (IEOM)*, 1–4.

G. D. Col and E. C. Teppan. 2017. "Learning Constraint Satisfaction Heuristics for Configuration Problems." In: *International Workshop on Configuration (ConfWS'17)*. IESEG, 8–11. ISBN: 978-2-9516606-2-5.

R. Cöster, A. Gustavsson, T. Olsson, and A. Rudström. 2002. "Enhancing web-based configuration with recommendations and cluster-based help." In: *AH'2002 Workshop on Recommendation and Personalization in eCommerce*. Málaga, Spain, 1–10.

L. De Raedt, A. Passerini, and S. Teso. 2018. "Learning constraints from examples." In: *32nd AAAI Conference on Artificial Intelligence* (AAAI'18/IAAI'18/EAAI'18). AAAI Press, New Orleans, Louisiana, USA, 7965–7970. ISBN: 978-1-57735-800-8.

S. Dev Gupta, B. Genc, and B. O'Sullivan. 2021. "Explanation in Constraint Satisfaction: A Survey." In: *30th International Joint Conference on Artificial Intelligence*. Ed. by Z.-H. Zhou, 4400–4407. doi:10.24963/ijcai.2021/601.

M. D. Ekstrand, J. T. Riedl, and J. A. Konstan. Feb. 2011. "Collaborative Filtering Recommender Systems." *Found. Trends Hum.-Comput. Interact.*, 4, 2, (Feb. 2011), 81–173. doi:10.1561/1100000009.

S. P. Erdeniz, A. Felfernig, M. Atas, T. N. T. Tran, M. Jeran, and M. Stettinger. 2017. "Cluster-Specific Heuristics for Constraint Solving." In: *Advances in Artificial Intelligence: From Theory to Practice*. Ed. by S. Benferhat, K. Tabia, and M. Ali. Springer International Publishing, Cham, 21–30.

S. P. Erdeniz, A. Felfernig, R. Samer, and M. Atas. 2019. "Matrix factorization based heuristics for constraint-based recommenders." In: *34th ACM/SIGAPP Symposium on Applied Computing* (SAC '19). ACM, Limassol, Cyprus, 1655–1662. ISBN: 9781450359337. doi:10.1145/3297280.3297441.

A. Falkner, A. Felfernig, and A. Haag. 2011. "Recommendation Technologies for Configurable Products." *AI Mag.*, 32, 3, 99–108. doi:10.1609/aimag.v32i3.2369.

A. Falkner, G. Friedrich, A. Haselböck, G. Schenner, and H. Schreiner. Dec. 2016. "Twenty-Five Years of Successful Application of Constraint Technologies at Siemens." *AI Mag.*, 37, 4, (Dec. 2016), 67–80. doi:10.1609/aimag.v37i4.2688.

A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, and E. Teppan. 2018. "Industrial Applications of Answer Set Programming." *Künstl Intell*, 32, 165–176. doi:10.1007/s13218-018-0548-6.

A. Falkner, A. Haselböck, G. Krames, G. Schenner, H. Schreiner, and R. Taupe. 2020. "Solver Requirements for Interactive Configuration." *Journal of Universal Computer Science*, 26, 3, 343–373.

A. Falkner, A. Ryabokon, G. Schenner, and K. Shchekotykhin. 2015. "OOASP: Connecting Object-Oriented and Logic Programming." In: *Logic Programming and Nonmonotonic Reasoning*. Ed. by F. Calimeri, G. Ianni, and M. Truszczynski. Springer, 332–345.

A. Felfernig, A. Falkner, and D. Benavides. 2024. *Feature Models – AI Driven Design, Analysis, and Applications*. Springer, Cham.

A. Felfernig, M. Schubert, and C. Zehentner. Feb. 2012. "An efficient diagnosis algorithm for inconsistent constraint sets." *Artif. Intell. Eng. Des. Anal. Manuf.*, 26, 1, (Feb. 2012), 53–62. doi:10.1017/S0890060411000011.

A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. 2004. "Consistency-based diagnosis of configuration knowledge bases." *Artif. Intell.*, 152, 2, 213–234. doi:10.1016/S0004-3702(03)00117-6.

A. Felfernig, V.-M. Le, A. Popescu, M. Uta, T. N. T. Tran, and M. Atas. 2021. "An Overview of Recommender Systems and Machine Learning in Feature Modeling and Configuration." In: *Conference for Variability Modelling of Software-Intensive Systems* Article 16. ACM, Krems, Austria, 8 pages. doi:10.1145/3442391.3442408.

A. Felfernig, M. Schubert, and S. Reiterer. 2013. "Personalized diagnosis for over-constrained problems." In: *23rd International Joint Conference on Artificial Intelligence* (IJCAI '13). AAAI Press, Beijing, China, 1990–1996. ISBN: 9781577356332.

E. Freuder and B. O'Sullivan. 2014. "Grand challenges for constraint programming." *Constraints*, 19, 150–162. doi:https://doi.org/10.1007/s10601-013-9155-1.

E. C. Freuder. 2024. "Conversational Modeling for Constraint Satisfaction." *AAAI Conference on Artificial Intelligence*, 38, 20, 22592–22597. doi:10.1609/aaai.v38i20.30268.

E. C. Freuder. 1997. "In Pursuit of the Holy Grail." *Constraints*, 2, 1, 57–61. doi:10.1023/A:1009749006768.

E. C. Freuder and B. O'Sullivan. 2001. "Generating Tradeoffs for Interactive Constraint-Based Configuration." In: *CP 2001* (LNCS). Ed. by T. Walsh. Vol. 2239. Springer, 590–594. doi:10.1007/3-540-45578-7_45.

J. A. Galindo, A. J. Dominguez, J. White, and D. Benavides. 2023. "Large Language Models to generate meaningful feature model instances." In: *27th ACM International Systems and Software Product Line Conference - Volume A* (SPLC '23). ACM, Tokyo, Japan, 15–26. ISBN: 9798400700910. doi:10.1145/3579027.3608973.

D. Garber, T. Burgstaller, A. Felfernig, V. Le, S. Lubos, T. Tran, and S. Polat-Erdeniz. 2023. "Collaborative Recommendation of Search Heuristics." In: *25th International Workshop on Configuration (ConfWS 2023)*. Vol. 3509. CEUR-WS.org, Malaga, Spain, 38–44.

I. Gibson, W. Yoke San, Y. Huang, H. Liu, W. Keong Ng, W. Lu, B. Song, and X. Li. 2008. "Automating knowledge acquisition for constraint-based product configuration." *Journal of Manufacturing Technology Management*, 19, 6, 744–754. doi:10.1108/17410380810888120.

A. Günter and R. Cunis. 1992. "Flexible Control in Expert Systems for Construction Tasks." *Applied Intelligence*, 2, 4, 369–385.

S. Gupta, B. Genc, and B. O'Sullivan. Aug. 2021. "Explanation in Constraint Satisfaction: A Survey." In: *30th Intl. Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Z.-H. Zhou. Survey Track. (Aug. 2021), 4400–4407. doi:10.24963/ijcai.2021/601.

A. Haag and S. Riemann. 2011. "Product configuration as decision support: The declarative paradigm in practice." *AIEDAM*, 25, 2, 131–142.

X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. 2020. "LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation." In: *43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '20). ACM, Virtual Event, China, 639–648. ISBN: 9781450380164. doi:10.1145/3397271.3401063.

J. L. Herlocker, J. A. Konstan, and J. Riedl. 2000. "Explaining collaborative filtering recommendations." In: *ACM Conference on Computer Supported Cooperative Work* (CSCW '00). ACM, Philadelphia, Pennsylvania, USA, 241–250. ISBN: 1581132220. doi:10.1145/358916.358995.

S. Hochreiter. Apr. 2022. "Toward a broad AI." *Communications of the ACM*, 65, 4, (Apr. 2022), 56–57.

J. Huang, S. Huang, S. K. Moghaddam, Y. Lu, G. Wang, Y. Yan, and X. Shi. 2024. "Deep Reinforcement Learning-Based Dynamic Reconfiguration Planning for Digital Twin-Driven Smart Manufacturing Systems With Reconfigurable Machine Tools." *IEEE Transactions on Industrial Informatics*, 20, 11, 13135–13146. doi:10.1109/TII.2024.3431095.

U. Junker. 2006. "Configuration." In: *Handbook of Constraint Programming*. Ed. by F. Rossi, P. van Beek, and T. Walsh. Elsevier, Amsterdam, The Netherlands, 837–873.

U. Junker. 2004. "QuickXPlain: preferred explanations and relaxations for over-constrained problems." In: *19th National Conference on Artifical Intelligence* (AAAI'04). AAAI Press, San Jose, California, 167–172. ISBN: 0262511835.

S. Kadioglu, P. P. Dakle, K. Uppuluri, R. Politi, P. Raghavan, S. Rallabandi, and R. Srinivasamurthy. 2024. "Ner4Opt: Named Entity Recognition for Optimization Modelling from Natural Language." *Constraints*, 29, 3–4, 261–299. doi:10.1007/s10601-024-09376-5.

T. N. Kipf and M. Welling. 2017. "Semi-Supervised Classification with Graph Convolutional Networks." In: *arXiv*. https://arxiv.org/abs/1609.02907 eprint: 1609.02907 (cs.LG).

P. Kogler, W. Chen, A. Falkner, A. Haselböck, and S. Wallner. 2024. "Configuration Copilot: Towards Integrating Large Language Models and Constraints." In: *International Workshop on Configuration (ConfWS'24)*. Vol. 3812. CEUR-WS.org, 101–110.

J. Kolodner. 1993. *Case-based reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN: 1558602372.

Y. Koren, R. Bell, and C. Volinsky. Aug. 2009. "Matrix Factorization Techniques for Recommender Systems." *Computer*, 42, 8, (Aug. 2009), 30–37. doi:10.1109/MC.2009.263.

F. Koriche, C. Lecoutre, A. Paparrizou, and H. Wattez. July 2022. "Best Heuristic Identification for Constraint Satisfaction." In: *31st International Joint Conference on Artificial Intelligence*. Ed. by L. D. Raedt. (July 2022), 1859–1865. doi:10.24963/ijcai.2022/258.

V.-M. Le, L. A. Feldgrill, and A. Felfernig. 2026. "Robust Lazy Conflict Detection via Multi-Conflict Extraction and Genetic Diversity Control." In: *40th AAAI Conference on Artificial Intelligence*. AAAI Press, 19208–19215. doi:10.1609/aaai.v40i23.38995.

H. Li. 2022. "Language models: past, present, and future." *Commun. ACM*, 65, 7, 56–63. doi:10.1145/3490443.

X. Li, Y. Wang, C. H. Wu, and D. Y. Mo. 2025. "Mapping customer needs to product technical specifications in quality function deployment using graphical convolutional networks." *Enterprise Information Systems*, 19, 7-8, 2510346. doi:10.1080/17517575.2025.2510346.

C. C. S. Liem and A. Panichella. 2020. "Oracle Issues in Machine Learning and Where to Find Them." In: (ICSEW'20). ACM, Seoul, Republic of Korea, 483–488. doi:10.1145/3387940.3391490.

D. Mailharro. 1998. "A classification and constraint-based framework for configuration." *Artif. Intell. Eng. Des. Anal. Manuf.*, 12, 4, 383–397.

T. Männistö, T. Soininen, J. Tiihonen, and R. Sulonen. 1999. "Framework and conceptual model for reconfiguration." English. In: *AAAI 99 workshop on configuration, Orlando, Florida, USA, July 1999*. Ed. by G. Friedrich and B. Faltrings. AAAI Press, United States.

T. Männistö and R. Sulonen. 1999. "Evolution of Schema and Individuals of Configurable Products." In: *Advances in Conceptual Modeling*. Ed. by P. P. Chen, D. W. Embley, J. Kouloumdjian, S. W. Liddle, and J. F. Roddick. Springer Berlin Heidelberg, Berlin, Heidelberg, 12–23. ISBN: 978-3-540-48054-9.

L. Marchezan, E. Rodrigues, W. K. G. Assunção, M. Bernardino, F. P. Basso, and J. Carbonell. 2022. "Software product line scoping: A systematic literature review." *Journal of Systems and Software*, 186, 111189. doi:https://doi.org/10.1016/j.jss.2021.111189.

D. L. McGuinness and J. R. Wright. 1998. "An Industrial Strength Description Logics-Based Configurator Platform." *IEEE Intelligent Systems*, 13, 4, 69–77. doi:10.1109/5254.708435.

Y. Mechqrane, C. Bessiere, and I. Elabbassi. 2024. "Using large language models to improve query-based constraint acquisition." In: *33rd International Joint Conference on Artificial Intelligence* (IJCAI '24) Article 212. Jeju, Korea, 1916–1925. ISBN: 978-1-956792-04-1. doi:10.24963/ijcai.2024/212.

K. Michailidis, D. Tsouros, and T. Guns. 2026. "DCP-Bench-Open: Evaluating LLMs for Constraint Modelling of Discrete Combinatorial Problems." In: *arxiv:2506.06052*. https://arxiv.org/abs/2506.06052.

K. Michailidis, D. Tsouros, and T. Guns. 2024. "Constraint Modelling with LLMs Using In-Context Learning." In: *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)* (Leibniz International Proceedings in Informatics (LIPIcs)). Ed. by P. Shaw. Vol. 307. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 20:1–20:27. doi:10.4230/LIPIcs.CP.2024.20.

N. Mirzadeh, F. Ricci, and M. Bansal. 2005. "Feature Selection Methods for Conversational Recommender Systems." In: *IEEE International Conference on E-Technology, e-Commerce and e-Service*. IEEE Computer Society, USA, 772–777. doi:10.1109/EEE.2005.75.

S. Mittal and F. Frayman. 1989. "Towards a generic model of configuraton tasks." In: *11th International Joint Conference on Artificial Intelligence - Volume 2* (IJCAI'89). Morgan Kaufmann Publishers Inc., Detroit, Michigan, 1395–1401.

A. Montazeralghaem, G. Tennenholtz, C. Boutilier, and O. Meshi. 2025. "Asking Clarifying Questions for Preference Elicitation With Large Language Models." In: *arXiv*. https://arxiv.org/abs/2510.12015 eprint: 2510.12015 (cs.AI).

S. Muggleton. 1991. "Inductive logic programming." *New Generation Computing*, 8, 295–318. doi:https://doi.org/10.1007/BF03037089.

T. Niederer, D. Schloss, and N. Christensen. 2022. "Designing Context-Aware Chatbots for Product Configuration." In: *Chatbot Research and Design: 6th International Workshop*. Springer-Verlag, Amsterdam, The Netherlands, 190–210. doi:10.1007/978-3-031-25581-6_12.

T. E. Nordlander, E. C. Freuder, and R. J. Wallace. 2007. "Maintaining constraint-based applications." In: *4th International Conference on Knowledge Capture* (K-CAP '07). ACM, Whistler, BC, Canada, 79–86. ISBN: 9781595936431. doi:10.1145/1298406.1298422.

B. O'Sullivan, A. Ferguson, and E. Freuder. 2004. "Boosting Constraint Satisfaction Using Decision Trees." In: *16th IEEE International Conference on Tools with Artificial Intelligence*, 646–651. doi:10.1109/ICTAI.2004.38.

R. Penco, D. Pintar, M. Vranić, and M. Šoštarić. 2025. "Large Language Model-Driven Framework for Automated Constraint Model Generation in Configuration Problems." *Applied Sciences*, 15, 12, 6518. doi:10.3390/app15126518.

J. A. Pereira, M. Acher, H. Martin, J.-M. Jézéquel, G. Botterweck, and A. Ventresque. 2021. "Learning software configuration spaces: A systematic literature review." *J. Syst. Softw.*, 182, C. doi:10.1016/j.jss.2021.111044.

A. Popescu, S. Polat-Erdeniz, A. Felfernig, M. Uta, M. Atas, V.-M. Le, K. Pilsl, M. Enzelsberger, and T. N. T. Tran. 2022. "An overview of machine learning techniques in constraint solving." *J. Intell. Inf. Syst.*, 58, 1, 91–118. doi:10.1007/s10844-021-00666-5.

S. D. Prestwich, E. C. Freuder, B. O'Sullivan, and D. Browne. 2021. "Classifier-based constraint acquisition." *Annals of Mathematics and AI*, 89, 7, 655–674. doi:10.1007/s10472-021-09736-4.

R. Reiter. Apr. 1987. "A theory of diagnosis from first principles." *Artif. Intell.*, 32, 1, (Apr. 1987), 57–95. doi:10.1016/0004-3702(87)90062-2.

H. Ren, H. Dai, B. Dai, X. Chen, D. Zhou, J. Leskovec, and D. Schuurmans. 2022. "SMORE: Knowledge Graph Completion and Multi-hop Reasoning in Massive Knowledge Graphs." In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (KDD '22). ACM, Washington DC, USA, 1472–1482. ISBN: 9781450393850. doi:10.1145/3534678.3539405.

F. Rossi, P. van Beek, and T. Walsh. 2006. *Handbook of Constraint Programming*. Elsevier Science Inc., USA. ISBN: 9780080463803.

F. Rossi and A. Sperduti. Oct. 2004. "Acquiring Both Constraint and Solution Preferences in Interactive Constraint Systems." *Constraints*, 9, 4, (Oct. 2004), 311–332. doi:10.1023/B:CONS.0000049206.43218.5f.

D. Sabin and R. Weigel. 1998. "Product Configuration Frameworks – A Survey." *IEEE Intelligent Systems*, 13, 4, 42–49. doi:10.1109/5254.708432.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. "The Graph Neural Network Model." *IEEE Transactions on Neural Networks*, 20, 1, 61–80. doi:10.1109/TNN.2008.2005605.

K. Shchekotykhin and G. Friedrich. 2009. "Argumentation Based Constraint Acquisition." In: *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*. Ed. by W. Wang, H. Kargupta, S. Ranka, P. S. Yu, and X. Wu. IEEE Computer Society, 476–482. doi:10.1109/ICDM.2009.62.

W. Shi, M. Liu, W. Zhang, L. Shi, F. Jia, F. Ma, and J. Zhang. 2025. "ConstraintLLM: A Neuro-Symbolic Framework for Industrial-Level Constraint Programming." In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Suzhou, China, 15999–16019. doi:10.18653/v1/2025.emnlp-main.809.

C. Sinz et al.. 2007. "Configuration." *IEEE Intelligent Systems*, 22, 1, 78–90. doi:10.1109/MIS.2007.6.

A. Smith, F. Greaves, and T. Panch. 2023. "Hallucination or Confabulation? Neuroanatomy as metaphor in Large Language Models." *PLOS Digital Health*, 2, 11, 1–3. doi:10.1371/journal.pdig.0000388.

W. Song, Z. Cao, J. Zhang, C. Xu, and A. Lim. 2022. "Learning variable ordering heuristics for solving Constraint Satisfaction Problems." *Engineering Applications of Artificial Intelligence*, 109, 104603. doi:https://doi.org/10.1016/j.engappai.2021.104603.

Y. Song and E. Cohen. 2026. "Do LLMs Understand Constraint Programming? Zero-Shot Constraint Programming Model Generation Using LLMs." In: *Learning and Intelligent Optimization*. Ed. by Y. Zhang, M. Hladik, and H. Moosaei. Springer Nature Switzerland, Cham, 16–31. ISBN: 978-3-032-09156-7.

M. Stumptner. 1997. "An overview of knowledge-based configuration." *AI Communications*, 10, 2, 111–125.

S. Szeider. 2025. "CP-Agent: Agentic Constraint Programming." In: https://arxiv.org/abs/2508.07468.

P. Temple, J. A. Galindo, M. Acher, and J.-M. Jézéquel. 2016. "Using machine learning to infer constraints for product lines." In: *20th International Systems and Software Product Line Conference* (SPLC '16). ACM, Beijing, China, 209–218. ISBN: 9781450340502. doi:10.1145/2934466.2934472.

D. Tsouros, H. Verhaeghe, S. Kadioglu, and T. Guns. 2023. "Holy Grail 2.0: From Natural Language to Constraint Models." In: *arXiv:2308.01589*.

D. Tsouros, S. Berden, and T. Guns. 2023. "Learning to Learn in Interactive Constraint Acquisition." In: *arXiv*. https://arxiv.org/abs/2312.10795.

T. Ulz, M. Schwarz, A. Felfernig, S. Haas, A. Shehadeh, S. Reiterer, and M. Stettinger. Aug. 2017. "Human computation for constraint-based recommenders." *J. Intell. Inf. Syst.*, 49, 1, (Aug. 2017), 37–57. doi:10.1007/s10844-016-0433-4.

M. Uta, A. Felfernig, D. Helic, and V.-M. Le. 2022. "Accuracy- and consistency-aware recommendation of configurations." In: *26th ACM Intl. Systems and Software Product Line Conference* (SPLC '22). ACM, Graz, Austria, 79–84. ISBN: 9781450394437. doi:10.1145/3546932.3546996.

M. Uta, V.-M. Le, A. Felfernig, and D. Helic. 2025. "Learning constraint orderings for direct diagnosis." *Journal of Intelligent Information Systems*, 63, 1753–1777. doi:https://doi.org/10.1007/s10844-025-00962-4.

S. Verma, V. Boonsanong, M. Hoang, K. Hines, J. Dickerson, and C. Shah. 2024. "Counterfactual Explanations and Algorithmic Recourses for Machine Learning: A Review." *ACM Comput. Surv.*, 56, 12, Article 312, 1–42. doi:10.1145/3677119.

H. Wang, H. Ren, and J. Leskovec. 2021. "Relational Message Passing for Knowledge Graph Completion." In: *27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (KDD '21). ACM, Virtual Event, Singapore, 1697–1707. ISBN: 9781450383325. doi:10.1145/3447548.3467247.

Y. Wang, X. Li, L. L. Zhang, and D. Mo. 2022. "Configuring products with natural language: a simple yet effective approach based on text embeddings and multilayer perceptron." *International Journal of Production Research*, 60, 17, 5394–5406. doi:10.1080/00207543.2021.1957508.

J. Wildstrom, P. Stone, E. Witchel, and M. Dahlin. 2007. "Machine learning for on-line hardware reconfiguration." In: *20th International Joint Conference on Artifical Intelligence* (IJCAI'07). Hyderabad, India, 1113–1118.

X. Xiang, S. Duan, H. Pan, P. Han, J. Cao, and C. Liu. 2021. "From One-hot Encoding to Privacy-preserving Synthetic Electronic Health Records Embedding." In: *International Conference on Cyberspace Innovation of Advanced Technologies*. ACM, Guangzhou, China, 407–413. ISBN: 9781450387828. doi:10.1145/3444370.3444605.

L. Zhang, X. Chen, A. Falkner, and C. Chu. 2014. "Open Configuration: a New Approach to Product Customization." In: *16th International Configuration Workshop, Novi Sad, Serbia, September 25-26, 2014* (CEUR Workshop Proceedings). Vol. 1220, 75–79.

Received 30 November 2025; accepted 20 May 2026