A Learning-Based Recommender System for Autotuning Design Flows of Industrial High-Performance Processors

Jihye Kwon
Department of Computer Science
Columbia University, New York, NY, USA
jihyekwon@cs.columbia.edu

Matthew M. Ziegler
IBM T. J. Watson Research Center
Yorktown Heights, NY, USA
zieglerm@us.ibm.com

Luca P. Carloni
Department of Computer Science
Columbia University, New York, NY, USA
luca@cs.columbia.edu

ABSTRACT

Logic synthesis and physical design (LSPD) tools automate complex design tasks previously performed by human designers. One time-consuming task that remains manual is configuring the LSPD flow parameters, which significantly impacts design results. To reduce the parameter-tuning effort, we propose an LSPD parameter recommender system that involves learning a collaborative prediction model through tensor decomposition and regression. Using a model trained with archived data from multiple state-of-the-art 14nm processors, we reduce the exploration cost while achieving comparable design quality. Furthermore, we demonstrate the transfer-learning properties of our approach by showing that this model can be successfully applied for 7nm designs.

1 INTRODUCTION

The main computing engine of any modern computer server is a high-performance processor realized as a very-large-scale integration (VLSI) circuit. State-of-the-art systems-on-chips host billions of transistors on a single chip [3, 5, 21]. Such outstanding computational capabilities have been enabled by the progress of computer-aided design (CAD) tools for *logic synthesis and physical design (LSPD)*, which have allowed hardware designers to cope with the remarkably growing complexity of VLSI design. Designers first specify the functionality of a circuit and then employ an *LSPD flow* to implement the circuit through a sequence of steps that include logic synthesis, physical placement, and routing. As most of these LSPD steps require the solution of many intractable (NP-hard) problems at a very large scale, it is infeasible for the CAD tools to automatically generate a final design that presents optimal values for multiple quality-of-result (QoR) metrics of interest, e.g., delay, power dissipation, routability, and area utilization. CAD tools also need to be flexible to handle various types of logic functionality, making it difficult to rely on a single algorithm for all designs.

These challenges have led advanced LSPD flows to provide a variety of *parameters* that affect the execution of CAD algorithms within the flow and, ultimately, impact the QoR. Table 1 provides an example of a few LSPD parameters from our own particular flow. As advanced LSPD flows may have 100s, or even 1000s of parameters, choosing parameters' settings can be extremely complex. Therefore, *tuning* LSPD flow parameters for each *macro* (a circuit partition[1]) is now one of the main tasks performed by professional logic and physical designers. While experienced designers may still manually tune parameters, automated parameter-tuning systems have recently emerged for LSPD and FPGA flows [7, 16, 23, 26]. Previously proposed systems for tuning CAD flows rely on iterative or adaptive online tuning algorithms that have significant runtime, diskspace, and total compute resource costs. More recently researchers have focused on reducing iteration counts by parallelizing the tuning algorithms: e.g. Xu et al. presented a distributed autotuning framework for optimizing FPGA designs [23], while Ziegler et al. presented a synthesis-parameter tuning system for VLSI designs that requires 3-5 iterations [26]. However, for industrial large-scale design efforts, there is a need to further reduce the parameter tuning costs. For instance, during a high-performance server design cycle, there may be 100s of macros that need to be individually tuned, and the CAD tool input data may change frequently.[2] Thus, even effective iterative tuning systems can stress industrial compute clusters and design schedules.

Table 1: Examples of LSPD *parameters*. These parameters were activated as a result of the DSE for an exemplary macro.

Parameter	Synthesis, placement, and optimization options
dpm	Area recovery and optimization after placement
fogs	Resizing late in the flow with accurate timing
latup	Allow upsizing of latches for timing
lpopt	Use specific low power optimization algorithm
sprd123	Spread out logic during optimization steps

For the next level of parameter tuning efficiency, we propose the concept of a learning-based recommender system that is specialized for VLSI circuit design. Recommender systems predict the affinity between each user and items, such as movies, music, or restaurants, to make personalized recommendations [8, 25]. The application of recommender systems to complex engineering tasks is a new and interesting avenue of research with a potential for significant impact. In the area of software engineering, several systems have been proposed to assist developers' activities [2, 19]. For our system, we consider the analogy where the LSPD flow parameter settings correspond to the items and the macros to the users.

Our proposed system consists of two modules: (1) an offline learning module, and (2) an online recommendation module. First, the offline learning module trains a collaborative QoR-prediction model based on tensor decomposition and regression. Then, the

[1] A macro is a separately synthesized component that is integrated into a larger chip.
[2] Industrial processor designs often follow synchronized LSPD version release schedules, e.g., a weekly or bi-weekly cadence.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
DOI: 10.1145/3316781.3323919

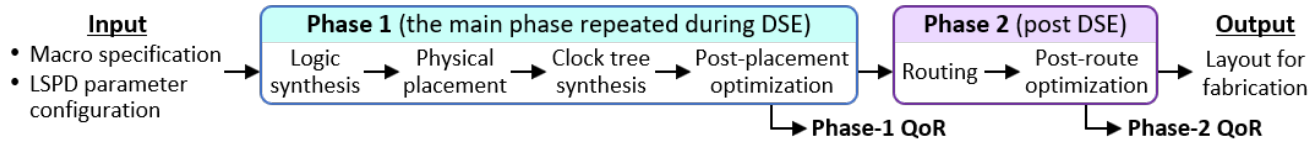

Figure 1: An LSPD flow deployed in an industrial environment for designing server-class high-performance processors.

online recommendation module employs the learned model to recommend the LSPD parameter configurations (the *scenarios*) for a given macro. The recommended scenarios can be used in various ways, e.g., 1) as a direct input to the LSPD flow, 2) directly combined with existing expert designer's settings for the specific macro, and 3) to provide suggestions for manual design modifications. The key contributions of this paper are summarized as follows:

- The first recommender system for VLSI design flow tuning.
- Novel algorithms for collaborative multi-metric QoR prediction based on tensor decomposition and regression.
- Results showing significantly improved QoR over default parameter settings for macros of high-performance processors.
- Best QoR achieved among the observed results when combining the recommended and designer's parameter settings.
- Demonstration that learning for 14nm designs can be transferred to new technologies, e.g., a 7nm design in progress.

2 RECOMMENDER SYSTEM OVERVIEW

LSPD Flow Overview. Fig. 1 shows a high-level diagram of an LSPD flow for industrial high-performance processors. The LSPD tool-chain takes as input a macro specification (RTL, timing requirements, physical boundary) and a configuration of LSPD parameters. The flow steps can be grouped into two main phases. The first phase performs a sequence of steps including logic synthesis, physical placement, clock tree synthesis, and post-placement optimization. Following these steps, key QoR metrics such as timing, power, and congestion (for routability) are reported based on estimated wire lengths, and these typically track well with the final QoR. Thus, designers often optimize QoR by tuning various LSPD parameters using Phase-1 QoR metrics as guidance in a process also referred to as *design-space exploration (DSE)*.[3] After the Phase-1 DSE, the second phase of the LSPD flow is executed to produce the final physical layout.

Our target LSPD tool-chain provides about 400 binary parameters, which act as meta-parameters. When one meta-parameter is activated (i.e. is set to *True*), a group of synthesis, placement, and/or optimization parameters are set to specific values. With 400 binary parameters, the design space of a macro corresponds to 2^{400} different physical implementations that could be synthesized for this macro. The goal of the DSE process is to locate one or more high-quality (*near-optimal*) parameter configurations.[4] For this DSE task we have employed an iterative parameter tuning flow to complete the design of multiple generations of industrial server-class processor chips.

As an example of iterative parameter tuning employed for prior processors, we consider the design of a double-precision floating-point pipeline macro. This macro contains 75,000 logic gates and takes 8 hours on average to be processed through the LSPD flow when deployed in an industrial environment targeting a 14nm semiconductor technology process (similar to the processes used in [3, 5, 21]). During 5 iterations of the parameter tuning process, 173 LSPD scenarios with different parameter configurations have been applied, i.e., each iteration included parallel execution of multiple scenarios. Table 1 reports the parameters of the top scenario determined through this process. While this iterative process has proven effective for many production processor designs, the overhead is still considerable, i.e., 173x compute resources and 5x runtime (latency). Reducing this overhead motivates our research.

LSPD Results Archive. The iterative tuning flow described in the previous subsection includes a background process that stores the data that are produced during each tuning run into an *archive*. The archive of LSPD results consists of the (*Input: macro, parameter configuration; Output: QoR*)–tuples from macros in multiple product families of high-performance processors, and over multiple design generations of these families. In total, the archive currently contains data from over 300,000 LSPD flow runs, from 1000s of macros across 22nm, 14nm, and 7nm technology nodes. We employ the archived LSPD results as training data for our proposed recommender system. Overall, the use of the iterative tuning flow provides an essentially free training set, whereas training sets for many other applications are curated through tedious and often manual processes. It should also be noted that the goal of the iterative tuning flow is to improve the QoR of the macro being tuned, not to supply a training set. Thus, the archived data is truly a by-product of the iterative DSE.

Once a macro is run through the iterative tuning flow and has data captured in the archive, we refer to it as a *legacy* macro, whereas we call a macro without archived data a *new* macro. In general, a new processor generation reuses (inherits) some logic from prior generations and contains legacy macros and new macros, i.e., a legacy macro reuses some amount of logic from a macro in prior generations. The amount of reuse can vary from nearly full logic reuse, e.g., when remapping to a new technology, to partial logic reuse, e.g., for a new architecture [6]. Other unobserved macros in the new generation processors are considered as new macros. When starting a new chip design, many macros are classified as legacy macros from observations in prior chips. Over the course of a chip design project, new macros are reclassified as legacy macros after they are iteratively tuned and added to the archive.

Collaborative Recommender System. Fig. 2 shows a high-level diagram of the proposed recommender system as well as

[3]For the target LSPD flow addressed in this paper, Phase-2 steps are computationally expensive, motivating the use of Phase-1 QoR metrics for DSE. Note that Phase-2 QoR metrics could also be used for DSE with lower Phase-2 overheads.

[4]We use the term *near-optimal* since the parameter tuning task is a black-box optimization problem and the design space is too large to perform an exhaustive search or to verify that an optimal solution is found. In practice, the goal of DSE is to locate solutions that provide notable improvements, rather than a precisely optimal solution.

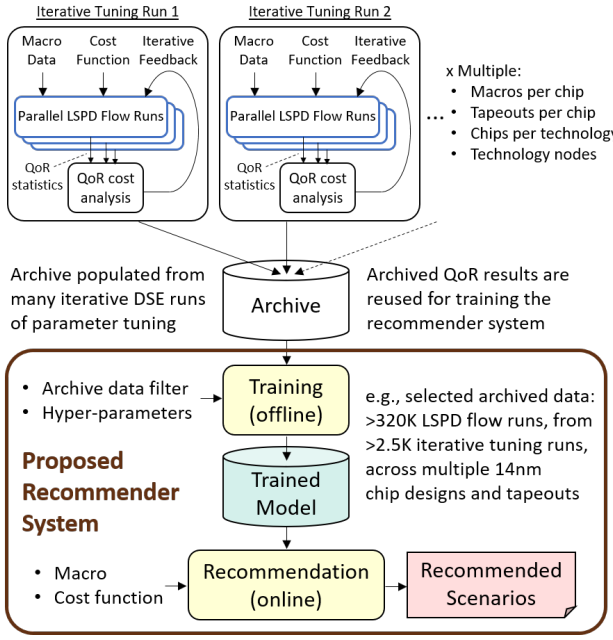

Figure 2: The recommender system is trained using archived data from multiple iterative tuning runs.

the interactions with the archive and iterative tuning flow. The proposed system consists of two modules: (1) the *offline learning module* and (2) the *online recommendation module*. The offline learning module trains a QoR prediction module using the LSPD results archive and the collaborative filtering approach. The online recommendation module takes as input the learned model, the target macro (the macro name for a legacy macro or sample LSPD results for a new macro), the QoR cost function or weight vector, and the number of scenario recommendations to generate. The module makes inferences using the given model and finally returns the requested number of recommended scenarios.

The recommender system's performance can be limited due to (1) the limited expressiveness of the model, (2) the sparsity of training data with respect to the huge search space, and (3) the complex nature of the problem, e.g., the existence of macro-specific or designer-specific parameters that the recommender system cannot address. In some cases, the performance can be improved by combining machine-generated scenarios with a design expert's input scenario (recorded in the archive for legacy macros). *The experimental results show that the collaboration between the recommender system and the design experts could lead to a QoR that is not achievable by either of them working solely.*

3 OFFLINE LEARNING MODULE

System Model and Problem Statement. Let $\mathscr{M}$ be a set of d macros: $\mathscr{M} = \{m_1, \cdots, m_d\}$, where m_i is a symbolic representation, e.g., the macro's name or index. Let $\mathscr{P}$ be a set of n binary (meta) parameters: $\mathscr{P} = \{p_1, \cdots, p_n\}$. A scenario s is a subset of $\mathscr{P}$, i.e., selected parameters that are set to *True*, while others are set to *False*. Then, we can define *QoR* as a function that maps a (*macro, scenario*)–pair to normalized QoR scores in ℓ metrics, presented as a real-valued ℓ-dimensional vector.[5]

$$QoR(m, s) = (q_1, \cdots, q_\ell) \in [0, 1]^\ell,\ m \in \mathscr{M},\ s \subseteq \mathscr{P}.$$

For each macro m, let $S(m)$ denote the set of all scenarios that were applied during the DSE for m. An archive $\mathscr{A}$ (shown in Fig. 2) contains the $(m, s, QoR(m, s))$–tuples for all $m \in \mathscr{M}$ and $s \in S(m)$.

The first target problem is to find a prediction model F that approximates the *QoR* function, where F also maps a (*macro, scenario*)–pair to an ℓ–dimensional vector.

Problem 1. Find a model F that minimizes

$$\sum_{m \in \mathscr{M},\ s \subseteq \mathscr{P}} ||QoR\,(m, s) - F\,(m, s)||^2.$$

In the above problem, the goal is to minimize the sum of L^2 distances between $QoR\,(m, s)$ and $F\,(m, s)$ for all macros m and scenarios s. However, for scenarios $s \notin S(m)$, the golden $QoR\,(m, s)$ values are unknown. Thus, we aim to minimize the distances between $QoR\,(m, s)$ and $F\,(m, s)$ only for the scenarios recorded in the archive $\mathscr{A}$, which acts as the training data for machine learning.

Problem 2. Given an archive $\mathscr{A}$, find a model F that minimizes

$$\sum_{(m, s, QoR(m,s)) \in \mathscr{A}} ||QoR\,(m, s) - F\,(m, s)||^2.$$

This can be viewed as a regression problem, attempting to predict the *QoR* values. One critical challenge is the lack of information regarding the input macro m and scenario s. A full specification of a macro is a collection of the designer's description, constraints, and linked libraries, that are neither easily available nor quantifiable.

Architecture of the Prediction Model. To address the aforementioned challenge, we exploit a collaborative filtering approach, which is widely used for recommender systems [8]. For instance, a movie recommender system recommends a new movie to a user, based on this user's rating for other movies, and all other users' ratings. Let matrix A represent the movie ratings by all users, where A_{ij} represents user i's rating on movie j. Then, without further information, the system can learn latent features of each user and each movie, by factorizing matrix A into a user matrix B and the transpose of a movie matrix C, i.e., $A = B \cdot C^{\mathrm{Tr}}$. This factorization can be done approximately when some elements of A are missing. After B and C are learned, a missing rating A_{ij} can be predicted as the (i, j)–element of $B \cdot C^{\mathrm{Tr}}$ [8].

Our proposed architecture of the prediction model is motivated by the above approach for movie recommender systems, but it differs in addressing the following additional challenges.

C1. Unlike movies, the observed scenarios are very sparse. In the iterative parameter tuning example in Sec. 2, only 173

[5] Unlike many other recommender systems where all users' ratings are given in a fixed range, e.g., from 1 to 5 stars, the evaluated QoR scores (e.g., timing, power) are distributed over different ranges of values depending on both the metrics and the macros. Let $s*$ be the most commonly applied scenario across all macros in the archive. We normalize the evaluated QoR score QoR^{ev}_k of the k-th metric ($1 \leqslant k \leqslant \ell$) for each macro m as follows: $QoR(m, s)_k = \frac{QoR(m,s)^{\mathrm{ev}}_k - \min_x QoR^{\mathrm{ev}}(m,x)_k}{\max_x QoR^{\mathrm{ev}}(m,x)_k - \min_x QoR^{\mathrm{ev}}(m,x)_k}$ if $s*$ has not been applied to macro m, and $QoR(m, s)_k = 0.5 + \frac{QoR^{\mathrm{ev}}(m,s)_k - QoR^{\mathrm{ev}}(m,s*)_k}{2\max\{\max_x QoR^{\mathrm{ev}}(m,x)_k - QoR^{\mathrm{ev}}(m,s*)_k, QoR^{\mathrm{ev}}(m,s*)_k - \min_x QoR^{\mathrm{ev}}(m,x)_k\}}$ if $s*$ has been applied to macro m. The resulting QoR scores are in the range of $[0, 1]$.

scenarios were observed for one macro, out of about 2^{400} scenarios. Moreover, sub-optimal scenarios for this macro were rarely observed while tuning parameters for other macros.

C2. While a movie rating prediction model outputs a single value for each (*user*, *movie*)–pair, the QoR prediction model outputs a vector with ℓ elements for each (*macro*, *scenario*)–pair.

To cope with **C1**, the prediction model factorizes the QoR information into a macro matrix, a parameter matrix, and the part that relates the latent information for (*macro*, *parameter*)–pairs to a *QoR* vector. On the other hand, **C2** indicates that ℓ individual models could be needed to predict each of the ℓ QoR metrics. Instead, we propose to construct one holistic model that predicts all ℓ metrics. This model can be described with a (*macro*, *parameter*, *metric*)–tensor in analogy to a (*user*, *movie*)–matrix. With this approach, we can reduce the number of variables describing the model,[6] and exploit all available information together to learn the latent features.

The proposed prediction model F describes the relationship between (*macro*, *scenario*)–pairs and their ℓ–dimensional *QoR* vectors by (1) a tensor decomposition approach for predicting missing values, and (2) an artificial neural network for the regression. Let T be a tensor whose (i, j, k)–element T_{ijk} represents an intermediate value of the k-th QoR metric for the (macro m_i, parameter p_j)–pair. These intermediate values, which are unknown at first, propagate through a neural network G that predicts the final *QoR* for a scenario. Thus, by the backward propagation of errors, the intermediate values can be adjusted, as well as other variables of G. Then, the tensor T containing the intermediate values can be decomposed into factor matrices containing the latent features.

Specifically, the tensor T has the shape of $|\mathscr{M}| \times (|\mathscr{P}| + 1) \times \ell$, where $|\mathscr{M}|$ is the number of archived macros, and $|\mathscr{P}| + 1$ is the number of parameters, including one pseudo-parameter that is always set to $True$. ℓ is the number of QoR metrics. By CP decomposition [20],[7] T is decomposed into the macro matrix M, parameter matrix P, QoR metric matrix Q, and one super-diagonal tensor of shape $h \times h \times h$, where h indicates the dimension of the latent features. The factor matrices M, P, and Q have dimensions of $|\mathscr{M}| \times h$, $(|\mathscr{P}| + 1) \times h$, and $\ell \times h$, respectively. By this decomposition, an (i, j, k)–element of tensor T can be computed as

$$T_{ijk} = \sum_{\alpha=1}^{h} M_{i\alpha} \cdot P_{j\alpha} \cdot Q_{k\alpha}. \tag{1}$$

Given elements of the tensor T, a single-layer perceptron network G predicts the final *QoR* vectors [22]. That is, G can be expressed by a coefficient matrix R, a bias vector b, and an activation function, e.g., $tanh$. For an input vector v, it returns $G(v) = tanh_{ew}(v \cdot R + b)$, where $tanh_{\text{ew}}$ denotes an element-wise $tanh$ function. Now, the input vector that corresponds to a (macro m_i, scenario s)–pair is defined as follows. For any parameter p_j, the vector $T_{ij:} = (T_{ij1}, \cdots, T_{ij\ell})$ represents intermediate QoR values for the (m_i, p_j)–pair. For notational simplicity, let $s(p)$ be 1 when the parameter p is in the scenario s, and 0 otherwise. The input vector is the concatenation of vectors $s(p_1) \cdot T_{i1:}, \cdots, s(p_n) \cdot T_{in:}$. Then, *QoR* for an (m_i, s)–pair can be predicted by $G(s(p_1) \cdot T_{i1:}, \cdots, s(p_n) \cdot T_{in:})$.

To summarize, the proposed model F is constructed as follows:

$$\begin{aligned} F(m_i, s) &= G(s(p_1) \cdot T_{i1:}, \cdots, s(p_n) \cdot T_{in:}) \\ &= tanh_{ew}((s(p_1) \cdot T_{i1:}, \cdots, s(p_n) \cdot T_{in:}) \cdot R + b) \end{aligned}$$

Since T can be described by the latent feature matrices M, P, and Q, the model F can be written as $F(m_i, s;\, M, P, Q, R, b)$. Here, F has two types of input: (1) the original input (macro m, scenario s) to F and (2) variables M, P, Q, R, b that describe how to compute F.

Training the Prediction Model. Training model F corresponds to learning its variables M, P, Q, R, and b. Let archives $\mathscr{A}$ and $\mathscr{B}$ contain the training and validation data, respectively. To solve the following problem, we use a stochastic gradient descent method [22]:

Problem 3. Given $\mathscr{A}$, find model F's variables that minimize

$$\sum_{(m,s,QoR(m,s))\in\mathscr{A}} ||QoR\,(m,s) - F\,(m,s)||^2 + \lambda_1 L^1(F) + \lambda_2 L^2(F).$$

To avoid overfitting F to $\mathscr{A}$, the L^1 and L^2 regularization terms $L^1(F)$ and $L^2(F)$ are added, each multiplied by small constants λ_1 and λ_2, respectively.[8] A trained model F is evaluated in terms of the validation error $\sum_{(m,s,QoR(m,s))\in\mathscr{B}} ||QoR\,(m,s) - F\,(m,s)||^2$. After a large number of training iterations, the offline learning module returns the model F with the smallest validation error.

4 ONLINE RECOMMENDATION MODULE

Given a trained QoR prediction model $F = F(m_i, s;\, M, P, Q, R, b)$, metric cost function or weights w, number t of scenarios, and target macro m, the online recommendation module returns t scenarios that are predicted to achieve near-optimal QoR scores (weighted by w) for m according to F.

For a legacy macro $m_i \in \mathscr{M}$, the $QoR(m_i, s)$ for any scenario s can be predicted by computing $F(m_i, s)$. *Making an inference using this model F takes much less time than applying an LSPD flow (e.g., a few minutes vs. a number of hours).*

For a new macro m^*, the recommendation module requires a number of sample LSPD results for this macro. In the case of a legacy macro $m_i \in \mathscr{M}$, the i–th row of the macro matrix M contains the latent features for this macro. Similarly, let μ denote the (unknown) latent feature vector for m^*. Then, $QoR(m^*, s)$ for a scenario s can be estimated by $F(m_1, s;\, \mu;\, P, Q, R, b)$. In this model, only the values of μ are unknown, since the values of P, Q, R, and b are included in the learned model F. Thus, this μ can be learned using the model F and a sample LSPD results archive C.

Problem 4. Given the model F and archive C, find μ that minimizes

$$\sum_{(m^*,s,QoR(m^*,s))\in C} ||QoR\,(m^*,s) - F(m_1, s;\, \mu;\, P, Q, R, b)||^2 + \beta.$$

β represents the sum of L^1 and L^2 regularization terms for μ. After μ is learned by the gradient descent method, the model F can be again used to make inferences for the new macro m^*.

[6] The term 'variables' is more commonly referred as 'parameters' or 'weights' in other machine learning applications and recommender systems. In this paper, we refer to them as 'variables' to avoid the confusion with 'LSPD parameters'.

[7] CP and Tucker are two widely used methods for tensor decomposition. Tucker is a generalization of CP, but CP allows us to interpret and manipulate the latent information for each macro separately.

[8] The L^1 regularization term is the sum of the absolute value of all variables v describing the model F, i.e., $L^1(F) = \sum_{v\in F} |v|$. Similarly, $L^2(F) = \sum_{v\in F} v^2$.

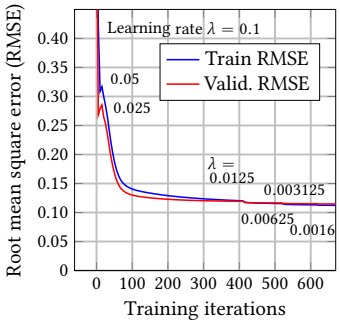

Figure 3: Train and validation error of the QoR prediction model.

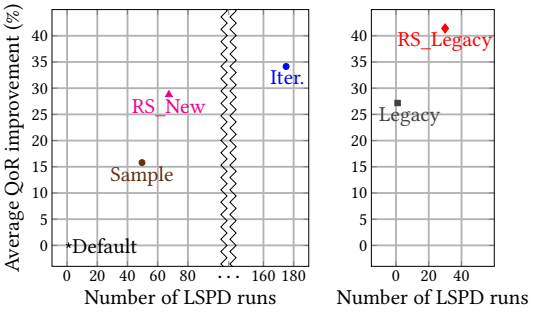

Figure 4: QoR comparison and the number of LSPD runs by the six methods.

Table 2: A representative set of five macros from industrial 14nm high-performance processors.

Macro name	Logic function	Logic gates	Runtime (Hrs)
FP	Floating-point pipeline	75K	8.0
ECDT	Exe. control & data transfer	45K	6.2
IDEC	Instr. decode	210K	21.6
ISC	Instr. sequencing control	77K	13.1
LSC	L2 cache control & FSM	195K	12.3

5 EXPERIMENTAL RESULTS

QoR Prediction Model. As described in Sec. 2, the LSPD results archive consists of the historical data for over 1,000 macros in multiple product families of high-performance processors, collected over a number of years and prior design efforts. For 250 binary meta LSPD parameters that are not too macro-specific or designer-specific, the archive contains about 300,000 (*Input: macro, scenario; Output: QoR*)–tuples, with 150,000 distinct scenarios.

We partition the archive into a train set $\mathcal{A}$ (80%) and validation set $\mathcal{B}$ (20%). The QoR prediction model F was trained by 600 iterations of the gradient descent method, with the variables initialized to random values between -0.5 and 0.5. The trained model's accuracy is evaluated by the root mean square error (RMSE) on the train set, i.e., *train RMSE* $= \sqrt{\sum_{(m,s)\in\mathcal{A}} ||F(m,s) - QoR(m,s)||^2/|\mathcal{A}|}$. The *validation RMSE* is defined similarly on the validation set $\mathcal{B}$. Fig. 3 shows the train and validation RMSE over the training iterations. Both RMSE values generally decrease as the number of iterations increases, with diminishing returns. The learning rate λ started from 0.1 and was decreased by a factor of 2 when the prediction accuracy (1−RMSE) did not improve over 10 iterations.

The dimension h of the latent feature vectors (in Equation (1)) was set to 50 to achieve a good balance between the capability of the model and the applicability to a new macro. With higher h, the model can express a more complicated function. However, to learn the latent features of a new macro by solving Problem 4 in Sec. 4, at least as many sample LSPD runs as h are needed.

LSPD Parameter Recommendations. To evaluate our recommender system, we selected five macros from a 14nm production processor. These macros, which perform distinct and critical logic functions, range in size from 45,000 to 210,000 logic gates (Table 2). The average LSPD (Phase 1) runtime varies from 6.2 to 21.6 hours.

QoR metrics of interest are the estimated worst slack, internal (register-to-register) slack, total negative slack, congestion score (routability), and total power. Since it is often very difficult to exactly close the timing for these macros during LSPD Phase 1, the goal of DSE has been set to minimize the weighted sum of QoR metrics, with a weight vector $w = [1, 2, 1, 3, 4]$. Fig. 4 shows the average improvement of $\max_s w \cdot QoR(m,s)$ and the number of LSPD runs for the five target macros, achieved by the following methods.

- *Default*: The default setting of LSPD, where all parameters are set to *False*. This does not result in the worst QoR and it is the baseline for comparing the QoR of other scenarios.
- *Sample*: About 50 parameters that were observed frequently and achieved high QoR according to the archive. The results from these scenarios are used by *Iterative* and *RS_New*.
- *Iterative*: DSE using a software program that iteratively improves the scenarios, following the approach proposed in [26].
- *RS_New*: 20 scenarios generated by the proposed method for a new macro, using results from 50 *Sample* runs.
- *Legacy (design expert's)*: The parameter configuration used by the designer who owned the macro for the production macro release. This may include settings of customized parameters.
- *RS_Legacy*: 30 scenarios recommended by the proposed method for a legacy macro, each combined (by set-union) with *Legacy*.

The five macros are considered as new macros for *Default*, *Sample*, *Iterative*, and *RS_New*, and as legacy macros for *Legacy* and *RS_Legacy*. Fig. 4 and the 14nm section (top) of Table 3 show results from this experiment. The overall best approach is *RS_Legacy*, which combines the recommended parameters for legacy macros with the design expert's configuration. *RS_Legacy* is the only approach that closes the timing with a positive internal slack (a key metric) for all five macros. It also outperforms other methods on the other two slack metrics and is close to the best in terms of power and congestion scores. While the *Iterative* method eventually achieves a high QoR improvement, it runs more than 170 LSPD scenarios, taking 5 iterations. Although the proposed method *RS_New* shows a slightly lower improvement, it takes about 70 LSPD runs on average, achieving higher efficiency (the slope from the origin to the improvement point on the chart) than the *Iterative* method. Moreover, many scenarios by *RS_New* or *RS_Legacy* have never been observed in the archive or by other methods.

In a second experiment, we explore transfer-learning capabilities of the recommender system by running a 7nm macro using scenarios recommended by a model trained with 14nm data [15]. For this experiment we use a 7nm version of the IDEC macro, which has similar logic functionality to the 14nm macro, but also some logic changes. The 7nm section (bottom) of Table 3 shows a comparison of the *Default* LSPD QoR and the *RS_Legacy14* approach, which is a combination of the recommended scenarios and the parameter configuration used for the final build of the 14nm version of the macro. *RS_Legacy14* provides a significant improvement in all three timing metrics, along with a small improvement in power and small degradation in congestion. Based on these results, we believe the recommender system will provide a solid starting point for new chip design projects in future technologies.

Table 3: Sum of QoR metrics over the five 14nm macros (top), and the metrics for a 7nm macro (bottom). Positive slacks, lower congestion and power are preferred.

		Worst slack (ps)	Internal slack (ps)	Total neg. slack (ps)	Cong. score (a.u.)	Total power (a.u.)
14nm	*Default*	-350	-288	-474,886	549	303
	Iterative	-195	-84	-126,774	441	253
	RS_New	-200	-110	-167,936	457	265
	Legacy	-202	-53	-89,675	516	278
	RS_Legacy	-130	15	-19,691	458	266
7nm	*Default*	-53	-52	-60,047	83	187
	RS_Legacy14	-10	-13	-4,384	86	184

6 RELATED WORK

Heuristic and machine learning methods for DSE. A large number of approaches have been introduced over the years, including using genetic algorithms for scheduling operations or binding resources during high-level synthesis (HLS) [4, 9]. Aine et al. proposed a profile-based iterative method to configure meta-parameters for CAD algorithms [1]. There are also a variety of methods based on the iterative refinements [7, 12, 14, 16, 23, 26]. More recent approaches based on machine learning include the transductive experimental design for sampling, combined with a random forest for predicting QoR of HLS runs [10], and simulated annealing, combined with a decision tree for reducing the HLS search space [11]. Also, Kapre et al. use a Bayesian approach to classify FPGA CAD parameter configurations based on their potential for timing gains [7]. Yanghua et al. perform the feature selection to reduce the number of FPGA CAD parameters to consider [24]. Instead of predicting the best (or Pareto-optimal) design points, Meng et al. eliminate the non optimal design points by regression [13]. With the exception of the approach proposed by Aine et al., all of the above methods perform the iteration or training for each design individually, while our approach explicitly leverages the information obtained during previous LSPD flow runs and makes macro-specific recommendations. In all of the above methods, a few 10s of parameters are considered for one DSE instance, whereas our approach can handle a few 100s of binary parameters.

Recommender systems. There are two main paradigms for recommender systems: content filtering and collaborative filtering [18]. The content filtering approach analyzes the content, and thus heavily depends on the availability and performance of the analysis methods [17]. On the other hand, the collaborative filtering approach exploits the collected information on how each user has reacted to each item. Since this approach mainly observes the users' reactions, it is fundamentally content-free and domain-free [8]. In this work, we propose a collaborative recommender system for LSPD parameters, since it is difficult to collect and analyze the individual macro specifications and LSPD parameters. Among collaborative filtering methods, the latent factor models decompose the users' affinity (the QoR in our case) into the latent properties of the items (LSPD parameters) and of the users (macros) [8]. Motivated by a variety of matrix factorization methods for these models, we propose a new latent factor model based on tensor decomposition [20], and an artificial neural network [22].

7 CONCLUSION

We present a recommender system of LSPD parameters, with the goal of reducing the high costs in VLSI design, especially for server-class high-performance processors. The proposed system learns the QoR prediction model using the LSPD archive and then uses it to generate scenario recommendations. In many cases, recommended scenarios are unique and have never been previously observed. Experimental results show that our approach can reduce the computational cost of DSE, and assist designers to improve the QoR by recommending scenarios to combine with their own configurations.

Acknowledgments. This work is partially supported by the NSF (A#: 1527821).

REFERENCES

[1] S. Aine et al. 2006. Improving the performance of CAD optimization algorithms using on-line meta-level control. In *Intl. Conf. on VLSI Design*. 683–688.
[2] B. Antunes et al. 2012. An approach to context-based recommendation in software development. In *Conf. on Recommender Systems*. 171–178.
[3] N. Beck et al. 2018. Zeppelin: An SoC for multichip architectures. In *Intl. Solid-State Circuits Conf.* 40–42.
[4] F. Ferrandi et al. 2008. A multi-objective genetic algorithm for design space exploration in high-level synthesis. In *Symposium on VLSI*. 417–422.
[5] C. Gonzalez et al. 2017. POWER9: A processor family optimized for cognitive computing with 25Gb/s accelerator links and 16Gb/s PCIe Gen4. In *Intl. Solid-State Circuits Conf.* 50–51.
[6] J. Hofmann et al. 2017. An analysis of core-and chip-level architectural features in four generations of Intel server processors. In *Intl. Supercomputing Conf.* 294–314.
[7] N. Kapre et al. 2015. Driving timing convergence of FPGA designs through machine learning and cloud computing. In *Intl. Symposium on Field-Programmable Custom Computing Machines*. 119–126.
[8] Y. Koren et al. 2009. Matrix factorization techniques for recommender systems. *IEEE Computer* 42, 8 (2009), 30–37.
[9] V. Krishnan and S. Katkoori. 2006. A genetic algorithm for the design space exploration of datapaths during high-level synthesis. *IEEE Tr. on Evolutionary Computation* 10, 3 (2006), 213–229.
[10] H. Liu and L. Carloni. 2013. On learning-based methods for design-space exploration with high-level synthesis. In *Design Automation Conf.* 50.
[11] A. Mahapatra and B. Schafer. 2014. Machine-learning based simulated annealer method for high level synthesis design space exploration. In *Electronic System Level Synthesis Conf.*
[12] G. Mariani et al. 2012. OSCAR: An optimization methodology exploiting spatial correlation in multicore design spaces. *IEEE Tr. on Computer-Aided Design of Integrated Circuits and Systems* 31, 5 (2012), 740–753.
[13] P. Meng et al. 2016. Adaptive threshold non-Pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs. In *Design, Automation & Test in Europe*. 918–923.
[14] G. Palermo et al. 2009. ReSPIR: a response surface-based Pareto iterative refinement for application-specific design space exploration. *IEEE Tr. on Computer-Aided Design of Integrated Circuits and Systems* 28, 12 (2009), 1816–1829.
[15] S. Pan et al. 2010. A survey on transfer learning. *IEEE Tr. on knowledge and data engineering* 22, 10 (2010), 1345–1359.
[16] M. Papamichael et al. 2015. Nautilus: Fast automated IP design space search using guided genetic algorithms. In *Design Automation Conf.* 43.
[17] M. Pazzani and D. Billsus. 2007. Content-based recommendation systems. In *The Adaptive Web*. Springer, 325–341.
[18] F. Ricci et al. 2011. Introduction to recommender systems handbook. In *Recommender Systems Handbook*. Springer, 1–35.
[19] M. Robillard et al. 2010. Recommendation systems for software engineering. *IEEE Software* 27, 4 (2010), 80–86.
[20] N. Sidiropoulos et al. 2017. Tensor decomposition for signal processing and machine learning. *IEEE Tr. on Signal Processing* 65, 13 (2017), 3551–3582.
[21] S. Tam et al. 2018. SkyLake-SP: A 14nm 28-Core xeon® processor. In *Intl. Solid-State Circuits Conf.* 34–36.
[22] B. Widrow and M. Lehr. 1990. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proc. IEEE* 78, 9 (1990), 1415–1442.
[23] C. Xu et al. 2017. A parallel bandit-based approach for autotuning FPGA compilation. In *Intl. Symposium on Field-Programmable Gate Arrays*. 157–166.
[24] Q. Yanghua et al. 2016. Boosting convergence of timing closure using feature selection in a learning-driven approach. In *Intl. Conf. on Field Programmable Logic and Applications*.
[25] F. Zhang et al. 2016. Exploiting dining preference for restaurant recommendation. In *Intl. Conf. on World Wide Web*. 725–735.
[26] M. Ziegler et al. 2016. A synthesis-parameter tuning system for autonomous design-space exploration. In *Design, Automation & Test in Europe*. 1148–1151.