CROP: Circuit Retrieval and Optimization with Parameter Guidance using LLMs

Jingyu Pan[1,†], Isaac Jacobson[1,†], Zheng Zhao[2], Tung-Chieh Chen[2], Guanglei Zhou[1], Chen-Chia Chang[1], Vineet Rashingkar[2], Yiran Chen[1]
[1]*Electrical and Computer Engineering, Duke University, Durham, USA*
[2]*Synopsys Inc., Sunnyvale, USA*
[1]{jingyu.pan, isaac.jacobson, guanglei.zhou, chenchia.chang, yiran.chen}@duke.edu
[2]{zheng.zhao, donnie.chen, vineet.rashingkar}@synopsys.com

arXiv:2507.02128v1 [cs.LG] 2 Jul 2025

***Abstract*—Modern very large-scale integration (VLSI) design requires the implementation of integrated circuits using electronic design automation (EDA) tools. Due to the complexity of EDA algorithms, the vast parameter space poses a huge challenge to chip design optimization, as the combination of even moderate numbers of parameters creates an enormous solution space to explore. Manual parameter selection remains industrial practice despite being excessively laborious and limited by expert experience. To address this issue, we present CROP, the first large language model (LLM)-powered automatic VLSI design flow tuning framework. Our approach includes: (1) a scalable methodology for transforming RTL source code into dense vector representations, (2) an embedding-based retrieval system for matching designs with semantically similar circuits, and (3) a retrieval-augmented generation (RAG)-enhanced LLM-guided parameter search system that constrains the search process with prior knowledge from similar designs. Experiment results demonstrate CROP's ability to achieve superior quality-of-results (QoR) with fewer iterations than existing approaches on industrial designs, including a 9.9% reduction in power consumption.**

***Index Terms*—design automation, parameter tuning, large language models**

I. Introduction

Modern industrial VLSI design flows exhibit exceptional complexity. A typical EDA flow comprises multiple sequential stages, with hundreds or even thousands of functions that depend on parameters which need to be configured. To optimize for an entire VLSI design flow, synthesis-related options in front-end tools like Synopsys Design Compiler or placement, clock tree synthesis, and routing options in back-end tools such as IC Compiler II must be configured appropriately to optimize for a specific QoR target [1]. The explorable parameter space grows exponentially with the number of configuration functions, making it practically impossible to exhaustively search for the best parameter combination given a chip design. Besides, the parameter configurations dramatically influence the QoR of VLSI designs across all metrics. For example, experiments showed that minor changes in synthesis parameters can lead to substantial variations in performance, power and area (PPA) [2]–[4].

Consequently, the automation of parameter tuning offers significant advantages for VLSI development. However, research in this field remains limited due to challenges in acquiring comprehensive design flow datasets necessary for implementing parameter tuning experiments. [5] developed an automatic parameter tuning approach using genetic algorithms that tests various parameter configurations. [6] leverage surrogate Gaussian process models to efficiently explore the design space by capturing correlations among training samples. [7] proposed Optuna, a general purpose hyperparameter optimization framework that allows dynamic construction of parameter search spaces during runtime. Recommender [8] trained a latent factor model with archived design data to recommend the best parameter configurations of a new design. [9] proposed a reinforcement learning-based method to suggest the best parameter configuration of a new design in one shot. FIST [2] automatically learns parameter importance from legacy designs and uses this information for feature-importance sampling and tree-based parameter tuning in new designs. However, these methods typically require a significant number of sample runs to identify optimal settings, as they lack any understanding of the circuit being optimized. The fundamental limitation lies in their lack of ability to leverage semantic understanding of the circuit being optimized, which leads to extra effort spent exploring regions of the parameter space that may be irrelevant for the target design.

TABLE I: Comparison of parameter tuning methods. CROP achieved circuit-aware and automatic parameter optimization with high efficiency.

Method	Circuit-Awareness	Automation	Efficiency
Human Effort	Yes	No	Low
Classical Algorithms [2], [10]–[12]	No	Yes	Low
CROP	Yes	Yes	High

To address this challenge, a promising research direction leverages circuit characteristics to enhance parameter tuning. This circuit-aware approach incorporates prior knowledge of design structure and functionality to more efficiently constrain the optimization process. VLSI design flows implementing such circuit-aware parameter tuning have demonstrated significant improvements in search efficiency. [13] used knowledge provided from circuit designers to guide the genetic algorithm and improve search efficiency. [11] leveraged a graph neural network and a transformer to help RL agent perform fast parameter exploration on unseen designs. [4] used random embedding to convert the parameters to a lower-dimensional latent space, which can be more efficiently explored based on design similarity. [10] exploited archival design data to improve the warm-up efficiency of the search engine. Despite these advances, significant challenges remain in effectively transferring knowledge in a way that better captures the correlation between pre-existing archival design data and the design currently being optimized. This is due to insufficient research on the effective encoding of the semantic representations of circuit designs. An approach that can understand circuit semantics and leverage knowledge from similar designs could dramatically reduce the number of iterations required to find optimal parameters.

Inspired by recent advances in LLMs and their application in EDA and chip design [14]–[16], we explore LLMs as a promising candidate for a fully-automated circuit-aware parameter search engine. Unlike traditional block-box optimization tools, LLMs can be conditioned

† Jingyu Pan and Isaac Jacobson contributed equally to this work.

using rich descriptive prompts that incorporate the QoR target, the functionality of each tool option, and even high-level descriptions of the circuit under optimization. This allows LLMs to emulate the intuition and practice of experienced engineers, making context-aware adjustments to EDA tool configurations, design constraints, and metric-oriented strategies. Furthermore, RAG emerges as a systematic approach that augments general-purpose LLMs with a domain-specific knowledge database, enabling more informed parameter optimization decisions based on historical design data and established optimization patterns. The integration of RAG with LLMs creates a parameter tuning framework that leverages contextual reasoning with circuit-specific prior knowledge, enabling parameter optimization that adapts to unique design characteristics across diverse technology nodes.

In this work we present **CROP**, the first LLM-powered automatic VLSI design flow tuning framework. Table I summarizes CROP's advantage compared with human effort and classical parameter tuning algorithms. The main contributions of this paper are listed as follows:

- A novel sub-framework for scalable summarization of a design from its RTL source code to a dense vector representation.
- An embedding-based design retrieval system to match designs with those that are semantically and structurally similar.
- A RAG-enhanced LLM guided automated search system that combines prior knowledge encoded in the LLM and that extracted from the design retrieval system.
- CROP finds better QoR than the baseline methods which even failed to explore the parameter space with better QoR values given a limited search budget. Specifically, CROP achieved a 9.9% reduction of the power consumption on an industrial processor design compared to the best of the baselines.

The remainder of this paper is organized as follows. Section II provides relevant preliminaries. Section III contains our problem formulation. Section IV illustrates our methodology. Section V presents experimental results. Section VI contains additional discussions. And Section VII presents our conclusion.

II. Preliminaries

A. Transformer-based Language Models

The advent of transformer-based language models has revolutionized the field of natural language processing (NLP) resulting in significant advancements in numerous tasks such as question answering, text generation, and machine translation. Introduced in 2017 by [17] the transformer architecture replaced recurrence with multihead self-attention, a mechanism that allowed more efficient training and better performance in sequence modeling tasks. Subsequent developments of larger models such as GPT3 [18] and LLaMA3 [19], have shown strong zero-shot and few-shot generalization skills when trained on massive corpora. These models are typically pre-trained for next-token generation and later finetuned for downstream tasks such as instruction following or question answering. Pre-trained transformer-based language models have achieved state-of-the-art accuracy on a wide range of NLP tasks and have shown significant general knowledge capabilities. More recently, new techniques have emerged to improve the reasoning capabilities of LLMs, such as Chain-of-Thought (CoT) prompting and reasoning architectures. CoT is a prompting technique that encourages step by step reasoning within a traditional LLM and has shown a significant improvement in complex reasoning tasks [20]. Reasoning models are specific architectures designed for reasoning often trained with reinforcement learning and using reasoning tokens to invoke an internal CoT-like process [21].

B. Retrieval-Augmented Generation

Retrieval-augmented generation, or RAG, is a popular class of methods to improve the quality of large language model inference by coupling text generation with a context-retrieval mechanism [22]. The retrieval mechanism queries external data sources, often databases or collections of documents, to extract relevant information. This additional information is added to the prompt before text generation is performed, which grounds responses with more up to date and comprehensive information [23]. A simple retrieval mechanism works by encoding the input query to create an embedding representation which is then used to try to match it to sections of the database that produce similar representations when encoded [24]. The generator is typically a sequence-to-sequence model (e.g., GPT) which conditions its output on both the input query and the additional retrieved context [22].

There are multiple advantages of RAG, including the ability of generative models to produce more factually accurate responses based on retrieved evidence that reduces the susceptibility of models to hallucinations [22]. Another major advantage is that the retrieved data allows a model to adapt to domains outside of its training data. The external database can be updated independently of the model's learned parameters, avoiding the need for expensive retraining or finetuning [23].

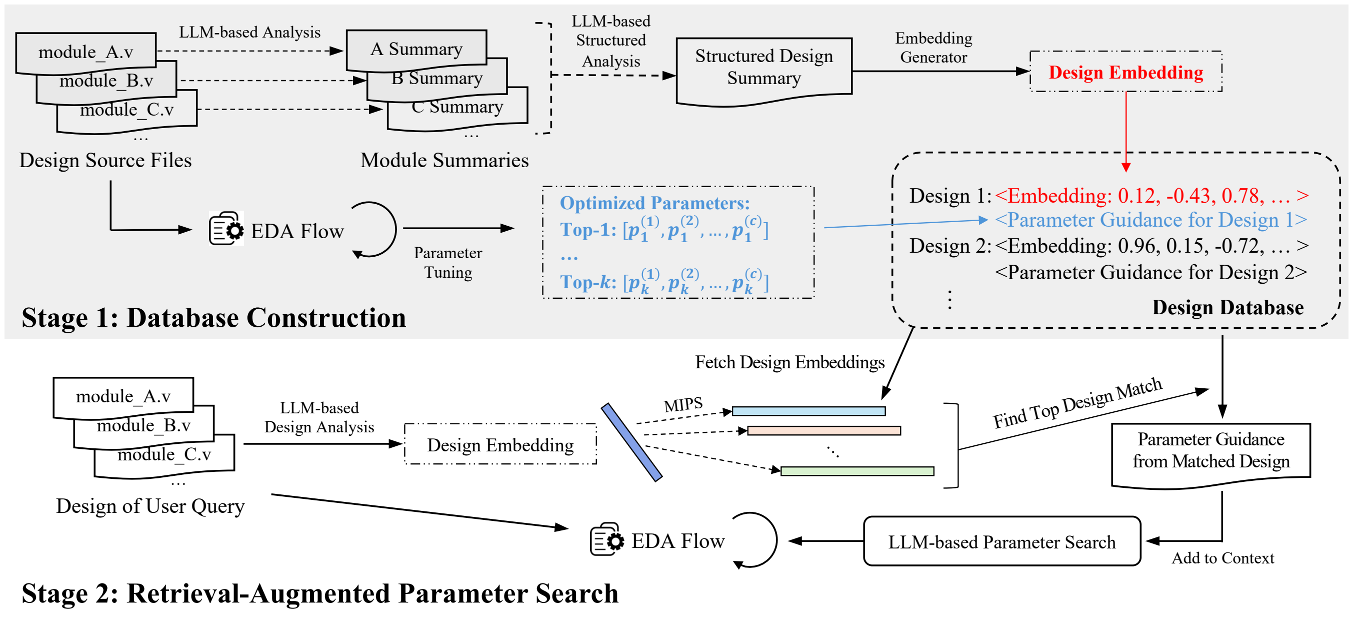
C. Parameter Tuning

Parameter tuning is a critical aspect of optimizing performance in a wide range of computational domains, from machine learning to scientific simulations. In general, it involves selecting values for a number of configurable parameters that affect the outcome, often trying to optimize for accuracy, execution time, or some other quantifiable metric.

In the context of EDA, parameter tuning plays a central role in optimizing a design's power usage, area, and performance or timing (PPA). Modern EDA tools expose a large number of configurable parameters, allowing experienced designers to control aspects of the synthesis, placement, and routing execution. The parameters include both categorical options, such as Boolean switches, and heuristic approach options, as well as numerical thresholds. In large, modern designs, it has been shown that different sets of parameters frequently lead to drastically different PPA results [5]. Traditionally, parameter tuning in EDA tools especially placement is a labor-intensive and time-consuming process relying on the expertise of human designers. Engineers often rely on trial and error approaches in which they iteratively adjust parameters based on the results they produce; this becomes extremely laborious and time-consuming as the EDA tools often take hours or even days to run each iteration [1], [2], [9], [10], [12]. Previous work in parameter tuning has shown that there are strong correlations between the most important parameters when optimizing similar designs for specific objectives [2]–[4], [25]. These works highlight that there are significant benefits to be gained from carrying over information regarding the tuning results between similar designs.

III. Problem Formulation

We refer to the configurable options in VLSI design flows as parameters. Each parameter combination is also referred to as a sample or a parameter vector. A parameter combination d consists of c parameters, and each parameter has n_i options, where $i \in [1, c]$. Continuous parameters can be discretized into categorical data. We use S to denote the whole parameter space and $|S| = \prod_{i=1}^{c} n_i$. The parameter space grows exponentially when c increases. Due to



module_A.v
module_B.v
module_C.v
Design Source Files
LLM-based Analysis
A Summary
B Summary
C Summary
Module Summaries
LLM-based Structured Analysis
Structured Design Summary
Embedding Generator
Design Embedding
EDA Flow
Parameter Tuning
Optimized Parameters:
Top-1: $[p_1^{(1)}, p_1^{(2)}, ..., p_1^{(c)}]$
Top-k: $[p_k^{(1)}, p_k^{(2)}, ..., p_k^{(c)}]$
Design 1: <Embedding: 0.12, -0.43, 0.78, … >
<Parameter Guidance for Design 1>
Design 2: <Embedding: 0.96, 0.15, -0.72, … >
<Parameter Guidance for Design 2>
Design Database
Stage 1: Database Construction
Fetch Design Embeddings
Design of User Query
LLM-based Design Analysis
MIPS
Find Top Design Match
Parameter Guidance from Matched Design
LLM-based Parameter Search
Add to Context
Stage 2: Retrieval-Augmented Parameter Search

Fig. 1: Overview of CROP. In stage 1, CROP constructs a design database. CROP first repeats the LLM-based design analysis to generate the design embedding. CROP selects the top-k parameter sets and their QoR results from existing optimization data to compile the parameter guidance for pre-existing circuits. In stage 2, CROP performs LLM-based analysis on the design of user query and retrieves the parameter guidance of the design from the database with highest similarity based on maximum inner product search (MIPS). The retrieved parameter guidance is used to constrain the follow-up LLM-based parameter search engine that optimizes the QoR for the input design.

the large parameter space, limited computational resources, and time constraints, only a small subset $\tilde{S}$ of samples can complete design flows and be evaluated. The process of selecting samples to form $\tilde{S}$ is referred to as sampling. The number of trials allowed is denoted as budget b, where $|\tilde{S}| \leq b$.

Given an optimization objective, the goal of our parameter tuning framework F is to find the sample with optimal objective value O using no more than b samples. Assume learning model f is used during exploration, we have:

$$\tilde{S} = F(S, b, f), \tag{1}$$

$$F^* = \arg\min_F (\min O[\tilde{S}] - \min O[S]). \tag{2}$$

To enable efficient parameter space exploration, we define design embeddings $E(D)$ as dense vector representations that capture the semantic and structural characteristics of circuit designs. These embeddings facilitate similarity-based retrieval from a database of previous designs. Parameter guidance of a design D is a set of top-k parameter configurations that have shown best performance in terms of the defined optimization objective for this design.

IV. Methodology of CROP

A. Overview

CROP is an LLM-powered parameter optimization framework that relies on a circuit knowledge retrieval system. Figure 1 shows the overview of our proposed CROP method, which is composed of two stages. In stage 1, we construct the database for future knowledge retrieval by the following steps: 1) Process the RTL source files of existing circuits via LLMs to generate module-level analysis summary and then a final summary for the whole circuit; 2) Use an embedding model to generate the embedding of the design based given its final summary as input, and use the design embedding as the indexes in the database. 3) Compile the parameter guidance of the design from pre-existing optimization data. Specifically, we select the top-k sets of parameter combinations and their QoR values to construct the parameter guidance. We detail the LLM-based design analysis and embedding generation in subsection IV-B.

In stage 2, CROP is applied to an unseen design of user query. The unseen design is also processed via LLM-based design analysis and design embedding generation. The generated design embedding is then used to retrieve the parameter guidance of the design with the highest similarity in the database. The retrieved parameter guidance is then utilized by an LLM-based parameter search engine. We detail the embedding-based design retrieval method in subsection IV-C and the LLM-based parameter search engine in subsection IV-D.

B. Design Analysis using LLMs

The LLM-based design analysis aims to generate a high-level semantic design representation that can be used for a knowledge retrieval system. First, we prompt an LLM with special instructions to iteratively summarize each module in the RTL code, prompting the model for a structured description of the module; the entries for which we prompt are shown in Table II. Then, the module-level summaries are concatenated with a new system prompt that queries the model for a structured high-level summary of the circuit's semantic and structural characteristics. The structure of the high-level summary is shown in Table III. We prompt the model for a combination of specific traits such as design name and primary inputs/outputs as well as more open-ended features like key design characteristics and potential applications. This ensures that all of the most important circuit traits are captured as well as using the LLM's prior knowledge to capture less consistent or quantifiable aspects of each design. This high-level analysis is then passed through an embedding model to retrieve a dense vector representation of the design, which is passed to the RAG engine. The iterative summarization of each module is an

TABLE II: Module-Level Summary Structure in CROP.

Entry	Description
inputs	Hierarchical mapping of input signals for each module, including bit widths and purposes.
outputs	Hierarchical mapping of output signals for each module, including bit widths and purposes.
overall functionality	Concise description of what the module does and its primary purpose.
critical submodules	Analysis of key submodules, their functionality, and internal logic implementation.
design choices	Architectural decisions made in the implementation, including techniques used.
timing constraints	Critical paths and timing considerations that affect performance.

TABLE III: Final Design Summary Structure in CROP.

Entry	Description
design name	A concise, descriptive name for the overall design based on its functionality.
design functionality	A brief summary of the overall function of the complete circuit.
primary inputs	Comprehensive list of all main inputs to the design as a whole.
primary outputs	Comprehensive list of all main outputs from the design as a whole.
key design characteristics	Notable features, architecture choices, and structure in 2-3 sentences.
key modules and functionalities	Mapping of important modules to their specific functions described concisely.
module interactions	How the individual modules communicate and interact within the design.
design optimizations	Notable optimizations in the design that affect performance or efficiency.
potential applications	Possible use cases and applications for this design.
timing considerations	Critical timing paths or constraints that affect overall performance.

important first step as it overcomes the scaling issue where the raw RTL code for sufficiently large designs can be tokenized to longer than the LLM's sequence length. Further discussion of this scaling issue is provided in Section VI.

C. Embedding-based Design Retrieval

As shown in Figure 1, CROP constructs a database with embeddings and parameter guidance of pre-existing designs in stage 1. When a user presents a target design as shown in Stage 2, CROP performs the same LLM-based design analysis process to generate embedding for the target design. This target design embedding is then compared with each embedding in the database using MIPS, where the design embedding with the highest inner product is chosen. Since design embeddings represent circuit characteristics in a latent space, when two designs have embeddings with higher inner product values, they likely share more semantic similarities in their functionality and specifications. This similarity-based retrieval system connects the target design with the most promising parameter guidance, effectively narrowing down the subsequent search process performed by large language models.

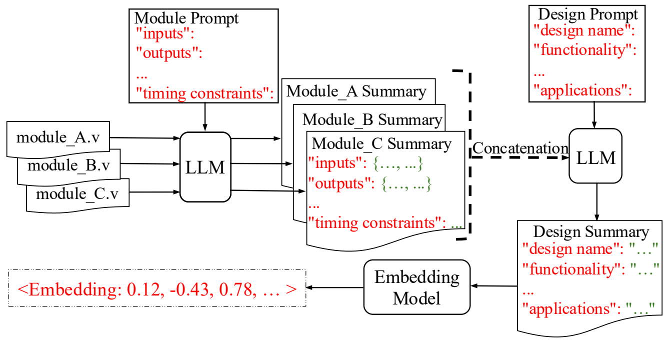


Fig. 2: Structure for LLM-based design analysis and embedding extraction.

D. RAG-enhanced LLM Search

The RAG-enhanced LLM search employs a decoder-only LLM as the parameter search engine. During each iteration, CROP constructs a prompt using general information, the parameter names and valid options, data retrieved through the RAG module, and the historical results of previous iterations. This prompt is provided to the model through an API which returns a response that is parsed for the suggested parameter sample. This sample is then validated and evaluated using RTL Architect, and the result is included in the prompts of future iterations. The general information consists of background and instructions. The background explains that the model is tuning EDA flow parameters for PPA objectives. The instructions outline the desired response format, as well as informing the model to be aware of both exploration and exploitation, fostering a more diverse coverage of the search space. We use a chain-of-thought model, as they have been shown to excel at tasks that require advanced scientific reasoning and domain-specific knowledge [21], [26]. This process allows an LLM to act as a fully autonomous design engineer leveraging prior knowledge while thoroughly exploring the search space, freeing human engineers from this laborious task.

V. Experiment Results

A. Experiment Setup

We evaluate CROP using a VLSI design flow based on Synopsys RTL Architect. We optimize 31 parameters related to placement optimization. Examples of these parameters are shown in Table IV. Each parameter corresponds to a configurable option of the RTL Architect flow. The parameters are selected to target coarse placement in the VLSI flow and are categorized into four groups for clarity: density, congestion, timing, and power. The parameters we use were selected based on industry best practices and have shown substantial performance improvements. Each parameter has a list of valid values for optimization algorithms to select, either as a categorical parameter or a continuous one which has been discretized into a set number of options. This gives us a fully discrete search space with more than 2.8×10^{15} possibilities. The size of the search space combined with the runtime of commercial EDA tools makes exhaustive search impossible. We employ the RTLRewriter benchmark [15] and two proprietary industrial processor designs to evaluate our framework. The RTLRewriter benchmark consists of a diverse collection of open-sourced RTL designs spanning various complexity levels and application domains. They provide a diverse range of designs on which we can evaluate the efficacy of our RAG engine. Additionally, we use two proprietary processor designs, which we refer to as *Prior Core* and *Target Core* to test the efficacy of CROP's ability to transfer

TABLE IV: Flow Parameter Examples for RTL Architect.

Category	Parameter	Value Range
Density	density_control_version	{1, 2, 3}
	enable_pin_density_aware	{true, false}
	enable_ultra_effort	{true, false}
	max_density	(0.0 - 1.0): 5 values
	max_density_adjustment	(0.0 - 1.0): 5 values
	max_pins_density	(1.0 - 1000000.0): 7 values
Congestion	congestion_level	(0.0 - 5.0): 10 values
	congestion_mode	{1, 2}
	congestion_style	{1, 2}
	congestion_version	{1, 2}
	enable_blockage_aware	{true, false}
	enable_congestion_aware	{true, false}
Timing	buffer_aware_mode	{0, 1, 2, 3, 4}
	clock_weight	{0, 1, 2, 3}
	delay_weight	(0.0 - 1.0): 5 values
	enable_buffer_aware	{true, false}
	enable_clock_aware	{true, false}
	enable_clock_latency_aware	{true, false}
	enable_clockgate_aware	{true, false}
	enable_clockgate_identification	{true, false}
	enable_clockgate_latency_aware	{true, false}
	enable_path_aware	{true, false}
	enable_registers_aware	{true, false}
	max_delay_weight	(1.0 - 10^{20}): 11 values
Power	power_mode	{0, 1, 2, 3, 4}
	power_weight	(0.0 - 1.0): 5 values
	power_weight_stdev	(0.0 - 100.0): 10 values

knowledge to new designs. Both cores consist of 111K cells and share similar architectural characteristics, but are implemented in different technology nodes, specifically, a 7 nm industrial tech node for *Prior Core* and 5 nm for *Target Core*.

We implemented three state-of-the-art baseline approaches for comparison on the same task: random search, Optuna, and Bayesian optimization (BO). As a naive baseline, random search selects each parameter value independently and without regard to previous outcomes. In each iteration, a new configuration is generated by randomly selecting an index for each parameter from its list of options using Python's built-in random module, resulting in a uniformly random distribution. Each trial is sampled i.i.d. from the search space, so this method does not utilize any historical information from previous evaluations. As another baseline, we used the Optuna hyperparameter optimization framework [7] to perform a guided search over the discrete search space. In our implementation, each of the 31 parameters is treated as a categorical variable and the optimizer makes a selection for each using `trial.suggest_categorical` which samples from the predefined set of values. We used Optuna's default sampling strategy, tree-structured parzen estimator, to control the search and manage the exploration-exploitation trade-off. The final baseline uses a Gaussian process-based optimizer implemented via the BayesianOptimization Python package [27]. This process models the optimization problem as a black box and fits a Gaussian process to the known points. Then it uses the Expected Improvement exploration strategy to select the next point of exploration [28]. We initialize this optimizer with ten points randomly sampled from the search space before commencing the Gaussian process, this ensures sufficient initial exploration. Throughout all experiments with CROP, we use an instance of GPT-4o for the design analysis, all-MiniLM-L6-v2 for text embedding of the design summary, and OpenAI o1 for the LLM-based search.

B. Circuit Retrieval Quality

The efficacy of a RAG framework for VLSI circuit optimization is inherently contingent upon the precision and relevance of the retrieval mechanism. The following experiments present an evaluation of our retrieval component, which constitutes the foundational element of the proposed framework. We quantitatively assess the retrieval quality through its retrieval precision, which measures the semantic alignment between query circuits and retrieved exemplars.

Figure 3 shows a t-SNE visualization of design description embeddings colored by each design's category, where individual points represent RTL module embedding vectors displayed with different colors and markers to indicate distinct design categories. The embeddings, generated from module source code and projected into 2D space using t-SNE, reveal clustering patterns that highlight both semantic similarities between designs within the same category and interesting cross-category relationships. For example, the alu, adder, multiplier, and subexpression circuits are all closely clustered, highlighting their shared relation to mathematical calculation (the subexpression circuits both implement a bitwise multiplier). Table V shows the precision scores for CROP's circuit retrieval mechanism on different families of designs. These results show an average precision score of 0.906 at $k = 1$ and 0.819 at $k = 3$, once again highlighting the ability of the design retrieval engine to successfully match designs with those that are semantically or functionally similar.

C. Parameter Tuning Performance

In these experiments we report power as the primary metric being optimized however, at the end of this section we present an analysis of the tradeoff between different PPA metrics. Figure 4 compares the performance of CROP against random search, BO and Optuna on *Target Core*. The results shown in Table VI demonstrate that CROP consistently outperforms all baseline methods across various search budget sizes. Specifically, CROP achieves a clear advantage over the baselines reaching 127.08 mW after only 5 iterations, and converges to a minimal power of 119.98 mW throughout the experiment. This accelerated convergence behavior suggests that the context enrichment from retrieving similar circuits (i.e., *Prior Core*) provides valuable guidance to effectively constrain the parameter search process. On the other hand, the lowest total power achieved by all other baselines after 75 iterations is 133.17 mW, which means that CROP achieved a **9.9%** reduction of the total power consumption compared to all baseline approaches. This substantial improvement

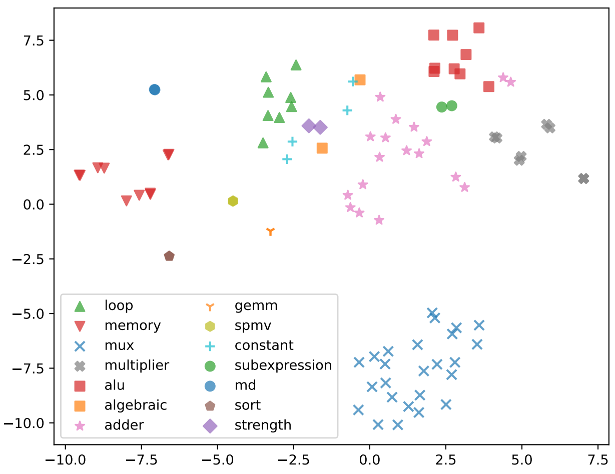


Fig. 3: A t-SNE-based visualization of design description embeddings colored by each design's category, designs from the RTLRewrite benchmark.

TABLE V: Retrieval Performance Evaluation of Circuit Embeddings.

(a) Average Precision and Recall at different k values

Metric @k	@1	@3	@5	@10	@20
Precision	0.906	0.819	0.767	0.651	0.506
Recall	0.193	0.331	0.441	0.636	0.852

(b) Precision@1 by Circuit Category

Category	Precision@1	Category	Precision@1
mux	1.000	md	1.000
memory	1.000	spmv	1.000
multiplier	1.000	alu	0.889
subexpression	1.000	adder	0.889
gemm	1.000	loop	0.875
strength	1.000	constant	0.250
sort	1.000	algebraic	0.000

(c) Dataset Statistics: 96 embeddings across 14 categories

Category	Count	Category	Count
mux	25	memory	10
adder	18	multiplier	8
alu	9	loop	8
constant	4	algebraic	2
gemm	2	md	2
sort	2	spmv	2
strength	2	subexpression	2

TABLE VI: Comparison of optimization algorithms on *Target Core* (Power in mW).

Method	5 Iterations	10	50	75
Optuna	134.63	134.63	133.45	133.24
BO	137.72	134.51	133.68	133.17
Random Search	135.52	135.11	133.44	133.44
CROP	**127.08**	**127.08**	**121.01**	**119.98**

highlights the efficacy of knowledge transfer between similar designs implemented in different technology nodes. Notably, while traditional optimization methods continue to make incremental improvements throughout their search trajectories, they fail to identify the high-quality parameter regions that CROP discovers almost immediately through its retrieval-augmented approach, demonstrating the limitations of design-agnostic search strategies for modern VLSI designs.

D. *RAG Benefit Analysis*

In order to evaluate the efficacy of the RAG component within CROP we compare CROP to a naive LLM-based search using the same model prompted without the information from the design retrieval component. We ran multiple iterations of both searches, which are shown in Figure 5. These results show a **7.5%** and **7.8%** reduction in power usage after 5 and 40 iterations, respectively, when including the RAG information in the search prompting. The performance of the naive LLM is comparable to those of the baseline methods shown in Figure 4 showing that LLM guided search on its own is a valid method for parameter tuning. This relies on the LLM's ability to intuitively understand the impact of parameters, tracking how they effect the result over time using only the knowledge encoded in the LLM's training. However, the RAG-aware LLM significantly outperforms any other method. The benefit of this RAG information is shown in how quickly it is able to find significantly better values than the other approaches, in some cases on the first iteration, as shown in the CROP_2 run. These results highlight both the efficacy of the RAG component in CROP's search and the ability of CROP to save significant amounts of time and computational resources.

E. *Parameter Importance Analysis*

Spearman's rank correlation coefficient [29] measures the strength and direction of monotonic relationships between two variables based on the ranked values rather than raw values. The coefficient ρ is calculated as $\rho = 1 - \frac{6\sum d_i^2}{n(n^2)-1}$, where d_i is the difference between the ranks of corresponding values and n is the number of value pairs. This metric quantifies parameter importance by assessing the monotonic association between parameter values and performance metrics without assuming linearity. Figure 6 compares the absolute Spearman's rank correlation coefficients between the top-impact parameters ($|\rho| > 0.10$) and power consumption across the two processor cores. We use absolute values to focus on the magnitude of each parameter's influence on power (i.e., its importance in terms of power optimization). Based on the absolute Spearman's ranking correlation data, we observe that high-importance parameters tend to be influential across both cores, suggesting underlying commonalities in design sensitivities. Although, some core-specific variations do exist such as `power_weight_stdev` (0.185, 0.018) which significantly impacts *Target Core* but not *Prior Core*, while `enable_clock_latency_aware` (0.018, 0.234) and `congestion_style` (0.006, 0.161) show the opposite pattern. The most influential parameters demonstrate strong importance across both cores, including `max_density` (0.372, 0.131), `power_mode` (0.180, 0.242), and `enable_congestion_aware` (0.199, 0.206). These are highly important parameters regardless of the current technology node. This supports the idea that optimized parameter sets from previous designs can be used to quickly understand the most important directions to explore within the search space accelerating convergence.

F. *Runtime Analysis*

As the overall goal of this work is to reduce the time parameter tuning takes, it is important to evaluate the runtime of our approach. Table VII shows the runtime breakdown for CROP as well as BO and Optuna. The runtime analysis of each of these methods reveals that running the EDA flow dominates the total runtime of the experiments.

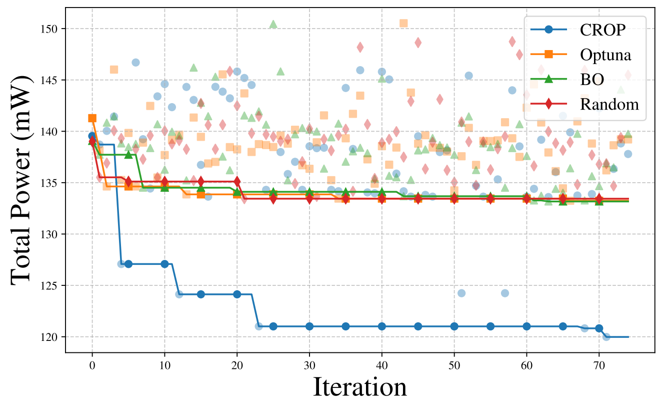


Fig. 4: Comparing the total power usage found by CROP and the baseline methods on *Target Core*.

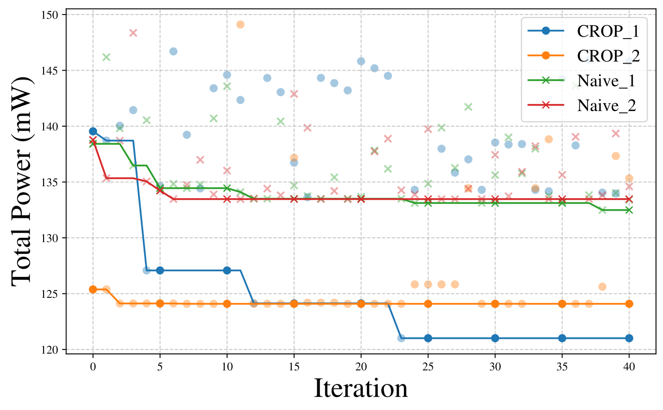


Fig. 5: Two iterations each of CROP and CROP without RAG information to show the advantage of RAG on *Target Core*.

CROP has additional initialization overhead from the RTL to embedding extraction and similarity-base retrieval mechanism, however, this overhead's impact is significantly reduced as more iterations are run. During each iteration CROP's parameter selection stage is mostly spent waiting on the inference of the reasoning model while Optuna and BO's are dominated by logging and file clean up with the actual next parameter calculation taking a small fraction of that time. Additionally, the baselines have negligible initialization overhead; however, the running of the EDA flow is no faster. The amount to which the EDA flow dominates the runtime of all approaches leads to very marginal differences between them. Automated approaches such as these will always be faster per iteration than those involving a human expert.

G. *PPA-tradeoff analysis*

The primary focus of the experiments presented early in this paper is power optimization. VLSI design is usually considered a multi-objective optimization problem where the design's power, area, timing, and number of design rule violations are all important. We thoroughly analyzed the correlation between these different values using the output of RTL Architect for both *Prior Core* and *Target Core* and concluded that power was an appropriate priority. Figure 7a shows that the designs' power and area results have a very strong positive correlation, meaning that the results with low power usage also have a small area. Figure 7b shows a positive correlation between the number of DRC violations and power values, but more importantly, all the lowest power values found have zero design rule violations. Overall *Prior Core* has far fewer DRC violations as a result of the fact that the newer technology nodes have more design rules to adhere to. When analyzing timing (in terms of total negative slack), we found a generally weak correlation between timing and power shown in Figure 7c. However, it should be noted that the LLM guided search on *Target Core* sacrificed timing to optimize power and area, this is shown by the orange dots in the bottom left corner. The lowest power point found by the search had a total of 3.88 ns of negative slack, while the most timing-aware points had only 2.54 ns of negative slack.

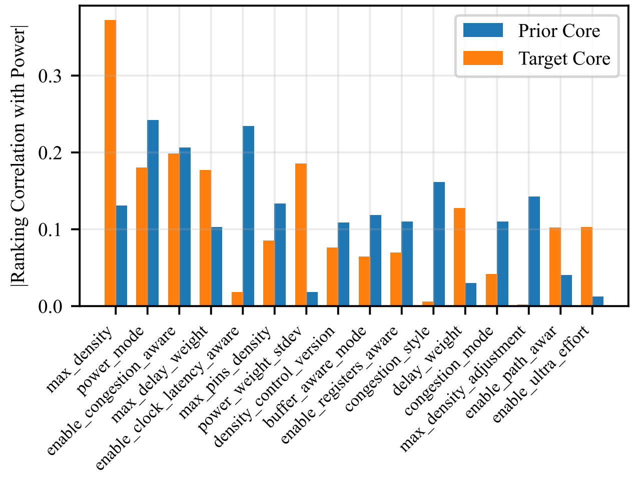


Fig. 6: The absolute Spearman's rank correlation between parameters and power consumption across target and prior processor cores.

TABLE VII: Runtime Breakdown by Framework Component for Optimizers (*Target Core*).

(a) CROP Runtime Analysis.

Component	1 Iteration	10 Iterations	50 Iterations
RAG Overhead	7.18%	0.77%	0.15%
Parameter Selection	0.99%	1.05%	1.06%
EDA Flow	91.83%	98.18%	98.79%

(b) BO Runtime Analysis.

Component	1 Iteration	10 Iterations	50 Iterations
Parameter Selection	0.14%	0.14%	0.14%
EDA Flow	99.86%	99.86%	99.86%

(c) Optuna Runtime Analysis.

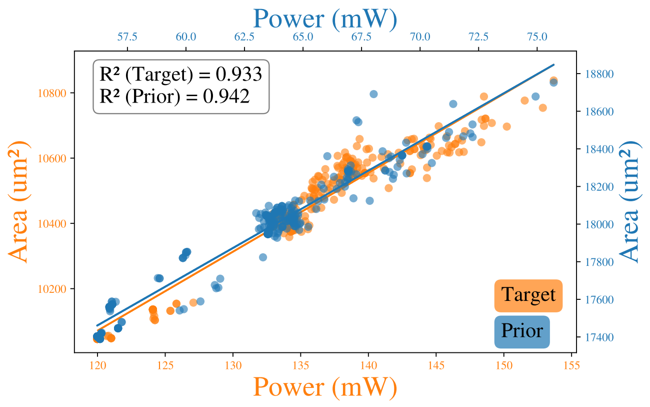
Component	1 Iteration	10 Iterations	50 Iterations
Parameter Selection	0.15%	0.15%	0.15%
EDA Flow	99.85%	99.85%	99.85%

VI. Discussion

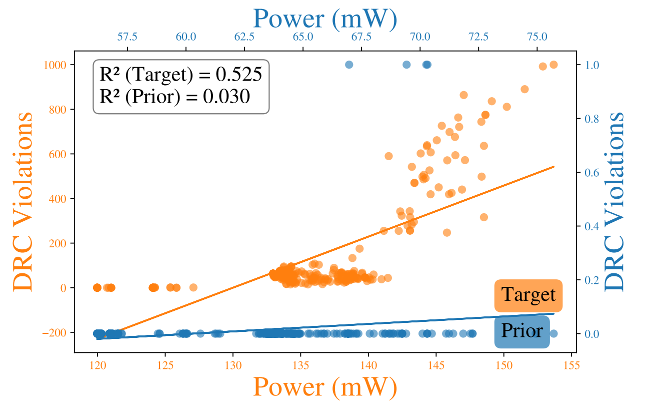
*a) **Addressing LLM scalability issue**:* The scalability of LLMs when processing RTL code is a fundamental challenge that our method addresses through an iterative summarization approach. The context window size of LLMs is crucial to their ability to craft coherent, accurate, and relevant responses for code analysis [30]–[32]. Industrial RTL codebases frequently exceed tens of thousands of lines, creating significant challenges for direct processing in LLMs due to the quadratic computational complexity of transformer attention mechanisms.

Our iterative summarization approach in Figure 2 overcomes this limitation by first processing individual modules separately and then synthesizing these summaries into a coherent design representation. This two-stage process effectively compresses the semantic content of large RTL codebases while preserving their functional characteristics. By generating embeddings from these structured summaries rather than raw RTL files, we maintain the essential design characteristics while dramatically reducing the token count required for processing. This approach also standardizes the representation format, making similarity comparisons more reliable across designs of varying complexity and size.

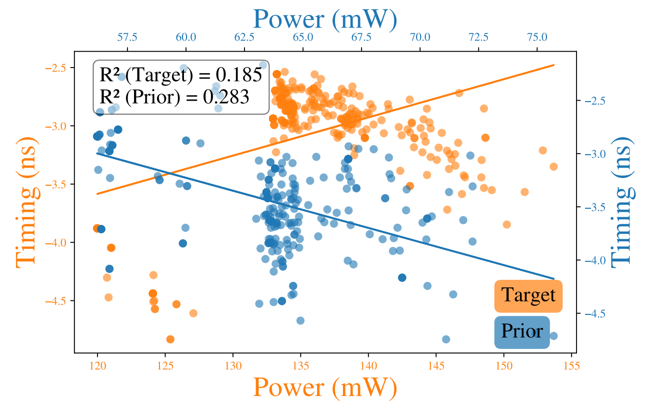
This method proves particularly valuable for industrial-scale designs where direct embedding of complete RTL code would exceed even the expanded context windows of modern LLMs (e.g., 128K



(a) Correlation analysis between power and area.



(b) Correlation analysis between power and number of DRC violations.



(c) Correlation analysis between power and timing in terms of negative slack.

Fig. 7: Correlation analysis between power and other QoR values for both *Prior Core* and *Target Core*.

tokens for GPT-4o in our experiment setting). Our experiments confirm that these derived embeddings successfully capture semantic similarities between functionally related circuits, as evidenced by the clustering patterns in Figure 3, while avoiding the prohibitive computational costs of processing entire RTL repositories directly.

b) ***Future directions****:* The experimental evaluations of CROP on industrial design show that it is a promising direction for improving automated flow tuning. The inherent database reliance of CROP is both a strength and a weakness; it requires similar designs to be present in the database in order to achieve best results but as more designs are ran through the framework they are added to the database expanding the capability of CROP. Furthermore, CROP is directly compatible with existing approaches that use jump-start or early-stopping methods. CROP is an EDA tool agnostic method that allows it to be used with any tool-chain, existing methods such as [4], [10] implement methods to reduce the runtime of the EDA tools. As shown in our run-time analysis, this has the potential to even further improve the overall efficiency of CROP. Additionally, CROP can be combined with search parallelization methods such as the creation of separate trust regions to search within. We hypothesize that this is a potentially highly beneficial direction of improvement as the results retrieved from the database can be used to reduce the number of initial iterations required to establish trust regions which can then be searched in parallel using multiple instances of LLM guided search.

VII. Conclusion

In this paper, we propose CROP, a novel framework for EDA tool parameter tuning based on the ability of an LLM to take the place of an experienced design engineer using prior knowledge extracted from a database of existing designs. We introduce a novel method for finding semantically similar designs using a dense vector representation of a design as well as a guided LLM based search, which is able to autonomously transfer prior knowledge from previous experiments. Our experimental results on industrial designs highlight that CROP is able to find better results in fewer iterations than existing approaches. Finally, future work is available combining CROP with existing methods to improve the runtime of EDA tools through approaches such as early stopping or parallelization.

Acknowledgment

This work is supported by NSF 2106828 and Synopsys Gift.

References

[1] H. Geng, T. Chen, Y. Ma, B. Zhu, and B. Yu, “Ptpt: Physical design tool parameter tuning via multi-objective bayesian optimization,” *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 42, no. 1, pp. 178–189, 2022.

[2] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany, S.-Y. Fang, J. Hu, Y. Chen, and E. C. Barboza, “Fist: A feature-importance sampling and tree-based method for automatic design flow parameter tuning,” in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 19–25.

[3] H. Geng, Q. Xu, T.-Y. Ho, and B. Yu, “Ppatuner: pareto-driven tool parameter auto-tuning in physical design via gaussian process transfer learning,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1237–1242. [Online]. Available: https://doi.org/10.1145/3489517.3530602

[4] S. Zheng, H. Geng, C. Bai, B. Yu, and M. D. Wong, “Boosting vlsi design flow parameter tuning with random embedding and multi-objective trust-region bayesian optimization,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 5, pp. 1–23, 2023.

[5] M. M. Ziegler, H.-Y. Liu, G. Gristede, B. Owens, R. Nigaglioni, and L. P. Carloni, “A synthesis-parameter tuning system for autonomous design-space exploration,” in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 1148–1151.

[6] Y. Ma, Z. Yu, and B. Yu, “Cad tool design space exploration via bayesian optimization,” in *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2019, pp. 1–6.

[7] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[8] J. Kwon, M. M. Ziegler, and L. P. Carloni, “A learning-based recommender system for autotuning design flows of industrial high-performance processors,” in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[9] A. Agnesina, K. Chang, and S. K. Lim, “Vlsi placement parameter optimization using deep reinforcement learning,” in *Proceedings of the 39th international conference on computer-aided design*, 2020, pp. 1–9.

[10] R. Liang, J. Jung, H. Xiang, L. Reddy, A. Lvov, J. Hu, and G.-J. Nam, “Flowtuner: A multi-stage eda flow tuner exploiting parameter knowledge transfer,” in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[11] H.-H. Hsiao, Y.-C. Lu, P. Vanna-Iampikul, and S. K. Lim, “Fasttuner: Transferable physical design parameter optimization using fast reinforcement learning,” in *Proceedings of the 2024 International Symposium on Physical Design*, 2024, pp. 93–101.

[12] P. Xu, S. Zheng, Y. Ye, C. Bai, S. Xu, H. Geng, T.-Y. Ho, and B. Yu, “Ranktuner: When design tool parameter tuning meets preference bayesian optimization,” in *2024 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2024.

[13] M. K. Papamichael, P. Milder, and J. C. Hoe, “Nautilus: Fast automated ip design space search using guided genetic algorithms,” in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[14] J. Pan, G. Zhou, C.-C. Chang, I. Jacobson, J. Hu, and Y. Chen, “A survey of research in large language models for electronic design automation,” *ACM Transactions on Design Automation of Electronic Systems*, 2025.

[15] X. Yao, Y. Wang, X. Li, Y. Lian, R. Chen, L. Chen, M. Yuan, H. Xu, and B. Yu, “Rtlrewriter: Methodologies for large models aided rtl code optimization,” in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–7.

[16] L. Chen, Y. Chen, Z. Chu, W. Fang, T.-Y. Ho, R. Huang, Y. Huang, S. Khan, M. Li, X. Li *et al.*, “The dawn of ai-native eda: Opportunities and challenges of large circuit models,” *arXiv preprint arXiv:2403.07257*, 2024.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS’17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[18] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[19] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.

[20] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS ’22, 2022.

[21] OpenAI, “Learning to reason with llms,” 2024. [Online]. Available: https://openai.com/index/learning-to-reason-with-llms/

[22] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS ’20, 2020.

[23] F. Petroni, A. Piktus, A. Fan, P. S. H. Lewis, M. Yazdani, N. D. Cao, J. Thorne, Y. Jernite, V. Plachouras, T. Rocktäschel, and S. Riedel, “KILT: a benchmark for knowledge intensive language tasks,” *CoRR*, vol. abs/2009.02252, 2020.

[24] V. Karpukhin, B. Oguz, S. Min, L. Wu, S. Edunov, D. Chen, and W. Yih, “Dense passage retrieval for open-domain question answering,” *CoRR*, vol. abs/2004.04906, 2020.

[25] Z. Zhang, T. Chen, J. Huang, and M. Zhang, “A fast parameter tuning framework via transfer learning and multi-objective bayesian optimization,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 133–138. [Online]. Available: https://doi.org/10.1145/3489517.3530430

[26] H. Cui, Z. Shamsi, G. Cheon, X. Ma, S. Li, M. Tikhanovskaya, P. C. Norgaard, N. Mudur, M. B. Plomecka, P. Raccuglia, Y. Bahri, V. V. Albert, P. Srinivasan, H. Pan, P. Faist, B. A. Rohr, M. J. Statt, D. Morris, D. Purves, E. Kleeman, R. Alcantara, M. Abraham, M. Mohammad, E. P. VanLee, C. Jiang, E. Dorfman, E.-A. Kim, M. Brenner, S. S. Ponda, and S. Venugopalan, “CURIE: Evaluating LLMs on multitask scientific long-context understanding and reasoning,” in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=jw2fC6REUB

[27] E. C. Garrido-Merchán and D. Hernández-Lobato, “Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes,” *Neurocomputing*, vol. 380, pp. 20–35, 2020.

[28] F. Nogueira, “Bayesian Optimization: Open source constrained global optimization tool for Python,” 2014–. [Online]. Available: https://github.com/bayesian-optimization/BayesianOptimization

[29] C. Spearman, “The proof and measurement of association between two things.” 1961.

[30] J. Li, M. Wang, Z. Zheng, and M. Zhang, “Loogle: Can long-context language models understand long contexts?” *arXiv preprint arXiv:2311.04939*, 2023.

[31] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, “A systematic evaluation of large language models of code,” in *Proceedings of the 6th ACM SIGPLAN international symposium on machine programming*, 2022, pp. 1–10.

[32] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, “Lost in the middle: How language models use long contexts,” *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.