Fine-tuning Large Language Models (LLMs) for AI-assisted Circuit Design and Simulation

***Abstract*—Circuit design and simulation constitutes the core task of any embedded-system development project. A circuit netlist file is fed to the circuit simulator for virtual prototyping and optimization tasks. In this work, a large set of circuit netlists in SPICE format are used to fine-tune existing large language models (LLMs) for generating circuit netlists from the user prompt. The re-trained LLMs are deployed at the back-end of the web-interface to generate the circuit netlists from the user-prompt. These circuit designs need to be imported to the simulator for further optimization. For that purpose, a user-friendly web-interface is developed using Django-framework with Ajax routines. Also, we have developed a user-friendly interface with our custom circuit simulator. The design goals of this circuit simulator and the interface with the LLM-based netlist generator are discussed.**

***Index Terms*—Circuit design and simulation, Large language models, natural language processing, AI-assisted netlist generation, Django-framework, AJAX.**

I. Introduction

Since the release of GPT 3.0 model by OpenAI in late 2020 [1], generative pre-trained transformer (GPT) based large language models (LLMs) have taken center-stage in the public discourse. This can be attributed partly to the capability of these models to accept natural language prompts as input and respond by giving articulate answers. Precision and humane nature of the answers has significantly improved over past years. Over the years, other LLMs have been developed by some of the major companies in software industry to enter into this rapidly evolving field.

LLM-based chat interface can also be viewed as a novel Human-Computer-Interface (HCI). It opens a new possibility in which a 'common-man' can also use specialized software without any prior training. Development of LLM-based digital design agents has been explored by [2]. Also, LLM-based design of specialized analog circuits has been shown by [3].

Historically, electronic circuit design has been a labor-intensive work. Due to the complexities involved, only a set of knowledgeable and skilled individuals were suitable for this task. However, the rapid penetration of electronic gadgets in daily life has opened a new field of application-specific gadgets. To serve this field, it is necessary to develop a rapid and reliable circuit designer from a generic user-prompt. The goal of our study is to retrain LLMs to generate circuit netlists from user prompts.

Since the existing users are most familiar with GUI-based interface, it is perhaps more suitable to develop LLM-based HCI in tandem with the existing GUI-based interface. Thus, our next goal is to find a suitable combination of LLM-based HCI and the GUI which can work together.

Goals of the paper are to,

1) Retrain or fine-tune various open-source LLMs to generate spice-based circuit netlists as per user prompts. Determine which LLM is more suitable for this purpose.
2) Develop an interface such that, both GUI and LLM-based netlist generator work in tandem to deliver the best of both worlds.

The paper is organized as follows. The step of fine-tuning the existing open-source LLMs is described in Section II. Inference, i.e. netlist generation using fine-tuned LLMs is discussed in Section III. Section IV and Section V explain, respectively, the web-interface and the LLM-interface with our circuit simulator `CircuitDraw` to assist designers in generating netlists.

II. Fine-tuning the LLMs

We used the following `Python` libraries to fine-tune the LLMs.

- `transformers` : Contains a class to load weights of open-source LLM - `AutoModelForCausalLM` . Also, provides `tokenizer` class which tokenizes input text.
- `bitsandbytes` : 4-bin and 8-bit quantization library. It converts 32-bit floating point weights into their 4-bit or 8-bit counterparts for faster and low-memory computation.
- `LoRa` : A parameter-efficient fine-tuning method for LLMs. It enables adaptation of a pre-trained LLM for a specific task with much lower computational overhead.
- `SFTTrainer` : Supervised Fine-Tuning (SFT) step is a common step in fine-tuning the models with field-specific data. We use `SFTTrainer` class for fine-tuning the models.

Effectiveness of a LLM depends largely on the the training dataset used for fine-tuning the models. We use the following two circuit netlist datasets-

- Vrindarani dataset - 106 netlists and their text description. [4]
- MasalaChai dataset - nearly 6000 netlists and their text description. [5]

The above datasets contain a list of `key−value` pairs. `key` is the textual description of the circuit, whereas `value` contains the circuit in SPICE netlist format. We convert these key-value pairs into a string of tokens in two steps.

1) Convert them to a python list of strings. Each string has the following format - `<s> [INST] <<SYS>>` You are circuit designer. `<</SYS>>` Design the circuit from the following description. `key [/INST] value`

TABLE I
NETLISTS GENERATED BY THE SELECTED FOUR VARIANTS OF LLMS FOR THE PROMPTS IN THE FIRST COLUMN.

Prompts	CodeLlama-13B		CodeLlama-7B	
	8-bit	4-bit	8-bit	4-bit

Generate a resistive voltage divider circuit. It contains two resistors in series. Middle net of the resistors is an output.	`V1 1 0 DC 5V`
	`R1 1 2 1k`
	`R2 2 3 1k`
	`R3 2 0 1k`
	`.END`
		`V1 1 0 DC 5V`
		`R1 1 2 1k`
		`R2 2 3 1k`
		`R3 2 0 1k`
		`Vout 3 0 DC 0V`
		`.end`
			`V1 1 0 DC 5V`
			`R1 1 2 1k`
			`R2 2 3 2k`
			`R3 3 0 3k`
			`.END`
				`V1 1 0 DC 5V`
				`R1 1 2 1k`
				`R2 2 3 2k`
				`R3 3 0 3k`
				`.END`

Generate an inductor based low pass filter. Do not use a capacitor.	`Vin 1 0 DC 0`
	`L1 1 2 100uH`
	`R1 2 0 10k`
	`Vout 2 0 DC 0`
	`.end`
		`Vin 1 0 DC 0`
		`L1 1 2 1uH`
		`L2 2 3 1uH`
		`L3 3 4 1uH`
		`L4 4 0 1uH`
		`.end`
			`V1 1 0 DC 0`
			`L1 1 2 100u`
			`R1 2 3 10k`
			`V2 3 0 DC 0`
			`.END`
				`V1 1 0 DC 0`
				`L1 1 2 100u`
				`R1 2 3 10k`
				`V2 3 0 DC 0`
				`.END`

Generate a band pass filter netlist. It contains a resistor, a capacitor, and an inductor, all in series.	`V1 3 0 DC 0`
	`R1 3 2 1k`
	`C1 2 0 1u`
	`L1 2 0 1m`
	`.END`
		`R1 2 3 1k`
		`C1 3 4 1u`
		`L1 4 0 1m`
		`.END`
			`V1 1 0 DC 0`
			`R1 1 2 1k`
			`C1 2 3 1u`
			`L1 3 0 1u`
			`.END`
				`V1 1 0 DC 0`
				`R1 1 2 1k`
				`C1 2 3 1u`
				`L1 3 0 1m`
				`.END`

Generate a amplifier using two stage BJT. Use BJT model of N40004.	`VCC 3 0 DC 15V`
	`Vin 1 0 AC 1V`
	`R1 3 2 1k`
	`Q1 2 1 4 N40004`
	`Q2 3 2 4 N40004`
	`.model N40004 NPN (IS=1E-14 BF =100)`
	`.end`
		`VCC 3 0 DC 15V`
		`Vin 1 0 AC 1V`
		`R1 3 2 1k`
		`R2 2 0 1k`
		`Q1 2 1 0 NPN`
		`.model NPN NPN (IS =1E-14 BF =100)`
		`.end`
			`VCC 3 0 DC 15V`
			`Vin 1 0 DC 0V`
			`R1 1 2 1k`
			`R2 2 3 1k`
			`Q1 2 2 0 N40004`
			`.model N40004 NPN`
			`.end`
				`Q1 2 3 4 N40004`
				`Q2 2 2 5 N40004`
				`VCC 2 0 DC 15V`
				`VEE 5 0 DC -15V`
				`* End of Netlist`

`</s>`. Here, `key` and `value` are the circuit description and the netlist, respectively.

2) Each of these strings is passed through the tokenizer to create numeric tokens.

In training phase, the above generated tokes are passed to the `SFTTrainer` to train the LLM. In the training phase, weights of the LLM are adjusted such that when the `key` is passed to the LLM, the corresponding `value` is returned.

"CodeLlama" LLM variant of Llama-models [6] is designed for code generation and analysis. Therefore, we believe it would be more suited for the structured text generation, such as netlists. We chose the two variants of "CodeLlama" LLM for fine-tuning.

- CodeLlama-7b-Instruct-hf : Contain 7 billion training parameters.
- CodeLlama-13b-Instruct-hf : Contain 13 billion training parameters.

Training all the parameters would be computationally intensive. Also, training them on the small size of the dataset would invariably lead to over-fitting of the model. To avoid it, only a sub-set of training parameters is used. The `LoRA` (low-rank-adaptation) module selects this sub-set of most relevant parameters in the LLM. We used the following LoRA modules - `gate_proj`, `down_proj`, `up_proj`, `q_proj`, `v_proj`, `k_proj`, `o_proj`. The chosen modules train 2.3% parameters out of 7 billion parameters of 'CodeLlama-7b' LLM. In case of 'CodeLlama-13b' LLM, the above modules select 1.8% parameters for training purpose. Value of some of the most relevant hyper-parameter used for the training step are given in Table II.

Using the LLM parameters at a lower precision is often sufficient to generate the desired output text. Using low precision parameters is also time-efficient and energy-efficient, when it comes to generating the netlists. In the training step, we have lowered precision of the LLM parameters by using `bitsandbytes` library for 4-bit and 8-bit quantization. As expected, 8-bit quantization uses higher precision than 4-bit quantization. Our aim is to study whether this difference in precision affects the netlist-generation quality.

Due to the high computational load of the LLM training step, training is performed by making use of hardware-accelerated (graphics-card) `transformers` library. Nvidia

TABLE II
PARAMETERS USED FOR FINE-TUNING THE LLM

Module	Parameter	value
SFT trainer	Training Epoch	4
	batch size	4
	optimizer	paged_ adamw_32bit
	learning rate	2×10^{-4}
	weight decay	1×10^{-3}
LoRA	alpha	16
	r	64
	dropout	0.1

TABLE III
TIME AND ERROR IN FINE-TUNING THE LLMS ON THE NVIDIA GTX-3090

LLM	Quantization	Training time	Training error
CodeLlama-7b	4-bit	5 Hr 30 min	0.649
	8-bit	6 Hr	0.65
CodeLlama-13b	4-bit	3 Hr 8 min	0.671
	8-bit	3 Hr 27 min	0.6739

graphics card "GTX3090" is chosen for the training step, due to 24Gb of video-RAM available on it. Additionally, presence of `bfloat16` based computational units on this graphics card allow faster computation (albeit at the cost of lower precision).

The LLMs in the list II were trained using the same set of parameters and on the same datasets to perform comparative analysis of their performance. Training time and the training error obtained at the end of the training steps are given in Table III.

III. INFERENCE USING FINE-TUNED LLMS

After the fine-tuning step, the fine-tuned LLMs were used to generate netlists by using a number of prompts. Same parameters as those used for training we also used for inference generation. We set the maximum number of additional characters to 128. The prompts were written in a way that models the user behaviour. Some of the prompts are given in the first column of Table I. In the table, the prompts are ordered as per the complexity of the desired circuit netlists. The generated netlists by each of the four variants of LLMs (CodeLlama 7B or 13B with 4-bit or 8-bit quantization) are shown in the second column of Table I. In total, 30 different prompts were passed to the LLMs to generate the netlists. We used the following criteria for testing usefulness of the generated netlists.

- Correctness of the spice format of the netlist,
- How closely the generated netlist resembles to the desired one,
- Whether the users can generate the desired circuit faster if they modify the generated netlist, instead of creating the desired netlist from scratch.

Generated netlists by each of the four variants of LLMs (CodeLlama 7B or 13B with 4-bit or 8-bit quantization) were tested per the above listed criteria. Our observations after testing the netlists are as follows.

1) All the variants generated netlists accurately as per spice format. The LLMs were able to reproduce the correct number of pins for for each of the components (e.g. 2-pins for R, L, and C, 4-pins for MOSFETs).
2) Among the four variants of the LLMs, CodeLlama 13B 8-bit quantization gave the most relevant answers, which resembled most to the desired netlist. 'CodeLlama-13B' has 13 billion parameters as opposed to '7B'. Also, 8-bit quantization uses these parameters with a higher precision, as opposed to 4-bit quantization. Therefore, better results are expected using 'CodeLlama-13B-8bit' model.
3) The LLMs can not set correct component values. For example, when asked to generate a passive RLC filter circuit of certain cut-off frequency, LLMs could not provide correct values for R, L, and C. Even the best of the LLMs confused between low-pass, band-pass, or high-pass filters!
4) LLMs could not generate netlists of some of the more complex circuits, e.g. fly-back-converter, buck/boost-converts. This is attributed to the absence of these complex circuits in training datasets.
5) We believe that, fine-tuned CodeLlama-13B with 8-bit can be useful for generating initial netlist, which can be further modified by the users.

Some more netlists generated by the best LLM among the four - *CodeLlama 13B 8-bit quantization* - are presented in

TABLE IV
NETLISTS GENERATED BY CODELLAMA-13B-8BIT FINE-TUNED MODEL.

Prompt	CodeLlama-13B-8bit

Generate an electric three phase motor driver circuit. It contains three MOSFET pairs. Output of each pair is connected to the motor.	`V1 1 0 DC 12V`
	`V2 3 0 DC 12V`
	`V3 5 0 DC 12V`
	`M1 2 1 4 4 NMOS`
	`M2 2 3 4 4 NMOS`
	`M3 2 5 4 4 NMOS`
	`R1 2 4 1k`
	`.END`

Create a full-bridge rectifier circuit which contains four diodes. Input is at 1 and output is at 3. Diode 1 is connected from 1 to 2. Diode 2 is from 2 to 3. Diode 3 is from 1 to 4. Diode 4 is from 4 to 3.	`V1 1 0 DC 0`
	`D1 1 2 Dmodel`
	`D2 2 3 Dmodel`
	`D3 1 4 Dmodel`
	`D4 4 3 Dmodel`
	`R1 2 3 1k`
	`.model Dmodel D`
	`.end`

Generate an analog current mirror circuit. This circuit acts as a driver to another current sensor.	`VDD 2 0 DC VDD`
	`IREF 2 3 DC IREF`
	`R1 2 3 R1_value`
	`R2 3 4 R2_value`
	`M1 3 3 0 0 NMOS_model`
	`M2 4 3 0 0 NMOS_model`
	`.model NMOS_model NMOS (Level =1)`
	`.end`

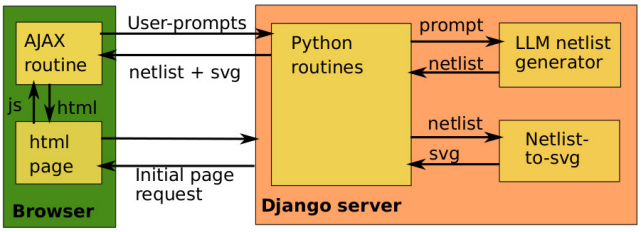

Fig. 1. Block diagram of the web-interface for netlist generation developed using Django-framework and AJAX routines.

Table IV along with the prompts used to generate them. Some of the complex circuits such as motor-drive circuit, were also generated fairly accurately. This shows capacity of the LLMs.

IV. Web Interface for Netlist Generation

Netlist generation is performed using the trained LLM using Python-based 'transformers'. The required high-end hardware acceleration is typically available only on a few machines. A 'client-server setup' allows for user-friendly hosting of the LLM on the server for netlist generation, and its use by multiple clients using the web-interface. Primary task of the web-interface is to receive natural language prompts from the users and provide generated netlists. For this task, we have developed Django server [7] which works as follows. User can type the prompt in the text-box displayed on the web-page. As soon as the user submits text prompt by clicking Send, *Ajax* script [8] runs at the client-side and collects all the previous user-prompts and server-responses. The collected prompts-response pairs are sent to Django server. The server submits the user-prompt to the LLM. The netlist generator generates spice-netlist and returns it to the client-side *Ajax* script for display. Block-diagram of the entire web-interface is shown in Fig. 1. Web interface is shown in Fig. 2.

A check-box Don't use history is displayed below the input-prompt box. If the check-box is un-checked, the previous prompt-response pairs are also sent to the LLM for 'context'. This 'context' allows for an interactive netlist generation. If the check-box is checked, the previous prompt-response pairs are ignored, and the circuit is generated solely from current user-prompt.

A. Circuit image auto-generation

It is often cumbersome to visualize the circuit by reading the netlist. To assist users with visualization, the server also renders circuit image of the netlist in svg format. After generating the netlist, we run ElkJs-based netlist-to-svg image converter on the server to generate the circuit svg image (see Fig. 2). It is then returned to the client-side *Ajax* script, which adds it to the HTML page. In this way, the svg image rendered in the chat window. We believe, that the svg image rendering would enable quick visualization of the circuit.

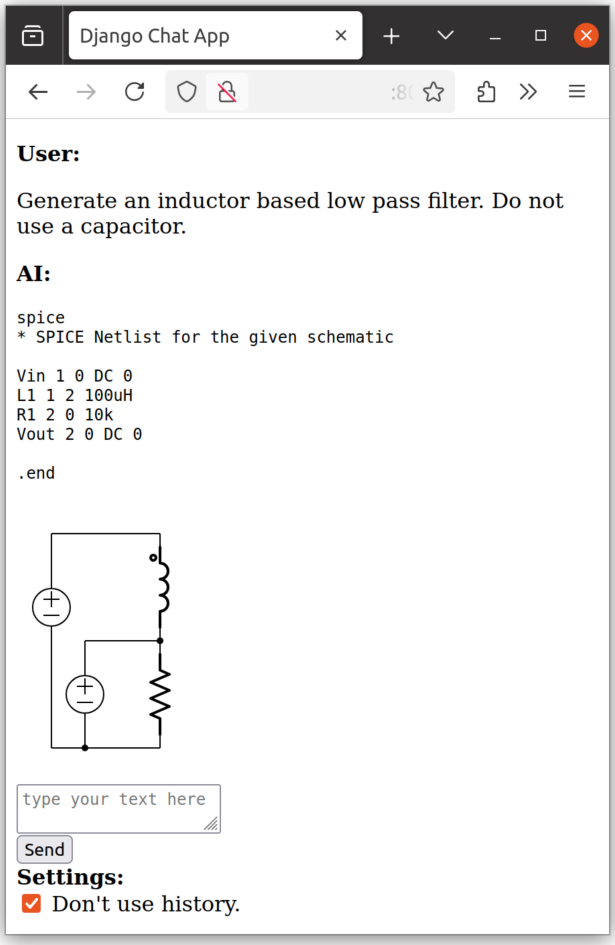

Fig. 2. Chat-like web-interface to generate netlist from the user prompts. An example prompt is shown in the figure. For visualization of the circuit, an svg image of the circuit is also created from the generated netlist and displayed below the chat.

V. Circuit Simulator - LLM Interface

A. Circuit Simulator GUI

GUI of a circuit drawing and simulation tool `CircuitDraw` by SemiVi [9] is shown in Fig. 3. The GUI window which consists of a 'Menu-bar', a 'Tool-bar', a circuit drawing board, and a side-pane. It enables interactive generation of electronic circuits and the circuit simulations using the `CircuitSolver` tool. The side-pane in the GUI named "Virtual AI Assistant" contains a web-engine for rendering the HTML page of the netlist generator. Users can write the prompt describing the circuit in the input-box and click `send`. The prompt along with the LLM-generated netlist are displayed on the webpage. If the netlist is a valid netlist, then SVG image of the netlist is also displayed below the netlist.

B. Interactive Circuit Generation

As seen in Fig. 3, the LLM-generated-netlist returned by the server is displayed in the side-pane along with the svg

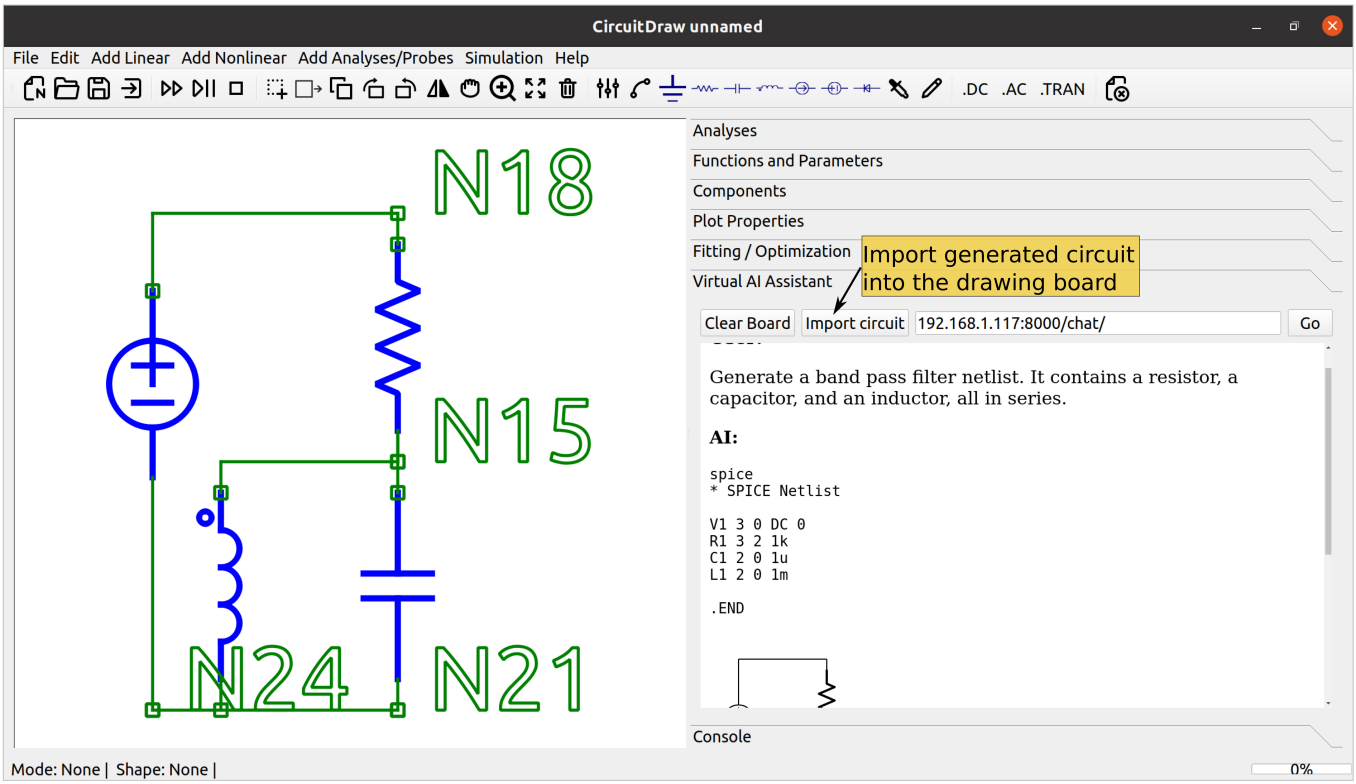

Fig. 3. CircuitDraw GUI together with the web-interface for LLM-based netlist-generator embedded in it.

image of the circuit. This netlist can act as a starting point for the subsequent circuit modifications. To refine the given netlist and adapt it for user-specific requirements, the netlist must be imported to the drawing board of 'CircuitDraw' tool.

The netlist is imported to the drawing board by clicking on Import circuit. Fig. 3 shows the circuit drawing board after clicking 'Import Circuit'. Once imported, the user can move the circuit components, add new components, change the component parameters, or add relevant analyses. In this way, the LLM-based netlist generation together with the `CircuitDraw` interface enables rapid circuit generation and prototyping.

VI. Conclusions

We have fine-tuned open-source LLMs, in particular, CodeLlama-7b and CodeLlama-13b to perform netlist generation, when a text prompt is submitted by the circuit designer. Two techniques, 4-bit and 8-bit quantization, were used for training the LLMs. The LLMs were successfully re-trained on the dataset of nearly 6100 netlist-text description pairs. After training, various text prompts were passed to generate netlists and assess their usefulness. 'CodeLlama-13b model' with 8-bit quantization was found to be more useful for netlist generation.

A chat-like web-interface using Django-framework and Ajax is developed to enable users submit text prompts and generate desired netlists. Also, the web-interface is embedded in the circuit drawing and simulation tool `CircuitDraw` for rapid circuit generation and simulation.

References

[1] T. Brown et al, "Language Models are Few-Shot Learners", Open AI, ArXiv, 2020, DOI: 10.48550/arXiv.2005.14165.

[2] C. Xiong et al.,"HLSPilot: LLM-based High-Level Synthesis", IEEE/ACM ICCAD'24, Article no. 226, pp. 1-9. 2024.

[3] J. Shen et al., "Atelier: An Automated Analog Circuit Design Framework via Multiple Large Language Model-Based Agents," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, doi: 10.1109/TCAD.2025.3573228.

[4] Vrindarani/netlistgen, Huggingface.

[5] J. Bhandari, V. Bhat, Y. He, H. Rahmani, S. Garg, and R. Karri, "Masala-CHAI: A Large-Scale SPICE Netlist Dataset for Analog Circuits by Harnessing AI," ArXiv, DOI: 10.48550/arXiv.2411.14299, 2025.

[6] H. Touvron et al, "LLaMA: Open and Efficient Foundation Language Models", arXiv, 2023. doi: 10.48550/arXiv.2302.13971.

[7] Django Software Foundation, 2019. Django, Available at: https://djangoproject.com.

[8] Ajax Frameworks, Apress, pp. 135–146, 2006. doi: 10.1007/978-1-4302-0183-0_7.

[9] Circuit Solver User Guide, SemiVi LLC, Switzerland, 2025.