I Kurzbericht CODAPE

Zuwendungsempfänger: **Fraunhofer Institut für Integrierte Systeme und Bauelementetechnologie (IISB)**

Projektleitung: **Dr. Andreas Rosskopf**

Verbund: **PE-Systems GmbH, BLOCK Transformatoren-Elektronik GmbH, Fraunhofer Institut für Integrierte Systeme und Bauelementetechnologie (IISB), Technische Universität Braunschweig - Institut für Elektrische Maschinen, Antriebe und Bahnen**

Thema: **CODAPE – Kollaborative Entwicklungsumgebung für die Leistungselektronik**

Dieser Bericht fasst die wissenschaftlich-technischen Arbeiten des Fraunhofer IISB der Jahre 2021 bis 2023 für das Projekt CODAPE zusammen.

Die übergeordnete Aufgabe bestand in der Konzeptionierung, Implementierung und Verifikation eines Optimierungsframeworks für die effiziente Abarbeitung von Schaltungs- und Simulationsjobs und die Bereitstellung dieser Funktionalität an die Projekt- und Industriepartner.

Zu Beginn lag der Schwerpunkt auf der Konzeptionierung eines Optimierungsframeworks zur effizienten Abarbeitung von über einer Million zeitlich stark variierenden Simulationsjobs. Das in Python entwickelte Framework, genannt „PyFlow“, ermöglicht die parallele Abarbeitung von Simulationsjobs und bietet (im Gegensatz zu damals verfügbaren Open-Source-Lösungen wie RAY) den Vorteil der direkten Integrierbarkeit von Optimierungsansätzen. Durch ein eigens eingebautes Scheduling-System konnten stark unterschiedliche Simulationsdauern und die Bearbeitung weiterer Simulationen in Abhängigkeit der aktuellen Ressourcenauslastung bei gleichzeitiger Beachtung definierbarer Ressourcengrenzen sichergestellt werden. Die Integration externer Anwendungen aus dem Bereich Schaltungssimulation und die Verwaltung von Simulationsaufträgen über standardisierte Konfigurationsdateien ohne anwendungsspezifische Code-Anpassungen wurden iterativ und unter Berücksichtigung des Nutzerfeedbacks (durch die Projektpartner) umgesetzt. Erste Tests und die Dokumentation der Implementierung wurden durchgeführt.

Zusätzlich zum reinen Load-Management wurden etablierte Genetische Algorithmen (GA) wie SPEA2 und NSGA2 - angepasst auf Fragestellungen der Schaltungsoptimierung - implementiert und in einem kaskadierten Ansatz im Interface „OptiFlow“ bereitgestellt. Die Integration von externen, während der Projektlaufzeit veröffentlichten Open-Source-Lösungen für die parallele Bearbeitung und Optimierung von Simulationsjobs wurden mittels passender Schnittstellen ermöglicht und nutzbar gemacht. Parallel zur Integration der GAs in das Simulationsframework erfolgten umfassende Tests, Verifikationen und viele Schleifen zur Anpassung der Hyperparameter (Wahrscheinlichkeiten und Art der Rekombination und Mutation) im Hinblick auf ingenieurswissenschaftliche Fragestellungen. Darauf aufbauend wurde das Framework in Hinblick auf hoch-parallele Simulationsausführungen mit Azure Cloud Ressourcen weiterentwickelt und mit den Projektpartnern getestet.

Parallel wurden Vergleiche und Benchmarks der GAs mit Monte-Carlo-Sampling Strategien durchgeführt um insbesondere für sehr schnelle Simulationen (>>1000 Samples / Sekunde) angepasste Vergleichsmetriken zu entwickeln. Basierend auf diesen Erkenntnissen wurde ein zusätzlicher Optimierungsansatz, die „Continuously Adapting Random Sampling“ (CARS) Methode, konzeptioniert und implementiert. Diese Methode fokussiert auf schnelle Simulationen und die Nutzung von hoch-parallelen Rechnersystemen und GPUs. Der Grundgedanke gründet auf einer Reinforcement Learning inspirierten Präferenz-Metrik, die „gute“ Parameter-Bereiche häufiger untersucht und „schlechte“ zunehmend ignoriert. Mittels verschiedener Erweiterungen und Steuerungsparameter gelang es Simulationsergebnisse auf benachbarte Parameterbereiche zu übertragen und damit wenig vielversprechende Parameterbereiche bei zukünftigen Simulationen zu vermeiden. Die CARS Methode

wurde für drei exemplarische Leistungselektronik-Anwendungen (Boost Converter und zwei LLC Resonanzwandler) mit GAs verglichen und zeigten eine signifikante Verbesserung der ermittelten gültigen Lösungskandidaten pro (Rechen-)Zeitintervall.

Darüber hinaus wurden im Rahmen des Projekts ein Demonstrator für eine digitale Entwicklungsumgebung für leistungselektronische Systeme entwickelt. Die Schnittstellen zu PE-Systems User-Interface und PE-Systems Simulationsumgebung wurden definiert und in Betrieb genommen. Eine API wurde implementiert, die mittels Python-Dictionaries Parameterwerte und Lösungskandidaten austauscht und in einem zweiten Schritt auch die Optimierungs-Settings (Art und Werte der Parameter, Nebenbedingungen und Zielfunktionen) definierte. Mittels dieser Schnittstellen konnten Systemoptimierungen effizient durchgeführt werden, während die beiden digitalen Lösungen – die effiziente Verwaltung und Optimierung der Simulationsdaten (Fraunhofer IISB), und Bereitstellung der Simulatoren (PE-Systems) - auf separaten Systemen arbeiteten.

Zur besseren Akzeptanz in der Ingenieurs-Community wurden Möglichkeiten zur Visualisierung von großen, hochdimensionalen Datensätzen untersucht. Dabei wurde ein existierender Open Source Ansatz von Facebook (HiPlot) getestet und angepasst, um die iterative Auswertung von Simulations- und Optimierungsergebnissen zu ermöglichen. Eine Erweiterung innerhalb einer Streamlit-Umgebung zur Darstellung von mehreren hunderttausend Datensätzen wurde damit umgesetzt und mit anwendungsspezifischen Funktionen ausgestattet.

Die Schnittstelle zum Projektpartner PE-Systems wurde zu Projektende ganzheitlich getestet und dokumentiert. Use Cases und Beispiele wurden erarbeitet und die „Pooling“- und „Oversampling“-Strategien wurden verfeinert. Mit den Projektpartnern wurden Best-Practice-Ansätze entwickelt und ein Benchmark zur Integration von einer Million Simulationsanalysen pro 10 Sekunden in das CARS-Framework durchgeführt. Ein modularer Workflow wurde aufgebaut, in dem CARS als ein eigenes Modul im Bereich „Optimization“ ausgewählt werden kann.
Für die Außendarstellung wurde ein Proof-of-Concept der API-Schnittstelle in dem Paper „Continuously Adapting Random Sampling (CARS) for Power Electronics Parameter Design“ ausgearbeitet und das Gesamtframework wurde im Rahmen eines Fachvortrags mit dem Titel „Continuously Adapting Random Sampling (CARS) for LLC Multi-Objective Parameter Optimization“ auf der ECCE2024 einem größeren Fachpublikum vorgestellt.

Zusammengefasst wurde im Rahmen des Projekts CODAPE vom Fraunhofer IISB ein leistungsfähiges Optimierungsframework zur, bisher unerreicht schnellen, parallelen Abarbeitung von Simulationsjobs entwickelt und unterschiedliche anwendungsspezifische Optimierungsmethoden integriert. Die CARS-Methode hat sich als effizienter Ansatz für schnelle Schaltungssimulationen erwiesen. Die Zusammenarbeit mit PE-Systems und die Nutzung von Open-Source-Lösungen haben zur Optimierung und Erweiterung des Frameworks beigetragen. Die durchgeführten Tests und Benchmarks sowie die Präsentation der Ergebnisse auf wissenschaftlichen Konferenzen unterstreichen den Erfolg des Projekts.

II Eingehende Darstellung CODAPE

Zuwendungsempfänger: **Fraunhofer Institut für Integrierte Systeme und Bauelementetechnologie (IISB)**

Projektleitung: **Dr. Andreas Rosskopf**

Verbund: **PE-Systems GmbH, BLOCK Transformatoren-Elektronik GmbH, Fraunhofer Institut für Integrierte Systeme und Bauelementetechnologie (IISB), Technische Universität Braunschweig - Institut für Elektrische Maschinen, Antriebe und Bahnen**

Thema: **CODAPE – Kollaborative Entwicklungsumgebung für die Leistungselektronik**

1. Aufgabenstellung und Voraussetzungen, unter denen das Vorhaben durchgeführt wurde

Im Rahmen des Förderprojekts CODAPE wird von zwei Industriepartnern (PE-Systems, BLOCK) und zwei akademischen Partnern (Fraunhofer IISB, TU Braunschweig), unterstützt durch insgesamt 9 assoziierte Partner, eine „Kollaborative Entwicklungsumgebung für die Leistungselektronik (LE)“ entwickelt. Insgesamt wurden dafür 209 Personenmonate (PM) eingeplant, wovon 24 PM auf den akademischen Partner Fraunhofer IISB entfallen.

	Arbeitspakete	PM	PE-Systems	BLOCK	Fraunhofer	TU Braunschweig
1	Definition und Anforderungen	13	6	3	2	2
2	Demonstrator für eine digitale Entwicklungsumgebung für leistungselektronische Systeme	35	20	9	3	3
3	Optimierungsframework	29	13	0	16	0
4	Optimierung Halbleiterdesign	26	25	0	0	1
5	Optimierung des Designs von Induktivitäten	25	4	21	0	0
6	Optimierung des Designs von Kondensatoren	19	16	0	3	0
7	Optimierung des Designs von Kühlkörpern	14	14	0	0	0
8	Design Verifikation	18	1	0	0	17
9	Softwaretest HMI und Einzelfunktionalität	13	7	6	0	0
10	Erprobung Demonstrator durch Entwicklung Schaltnetzteil (Verifikation Industrie)	17	4	12		1
		209	**110**	**51**	**24**	**24**

Die Arbeiten vom Fraunhofer IISB sind, in der TVB bzgl. fachlichen Gesichtspunkten sowie Aufwand, spezifiziert:

Das Fraunhofer IISB leitet die Konzeptionierung und Entwicklung des zentralen Optimierungsframeworks (**AP3: 18 PM**, Details siehe TVB) und unterstützt im Bereich „Definition und Anforderungen“ (**AP1: 2 PM**) sowie bei den Schnittstellen zur „digitalen Entwicklungsumgebung“ (**AP2: 3 PM**) und bei der Optimierung des Designs von Kondensatoren (**AP6:3 PM**).

Diese ursprünglich geplante Struktur konnte insgesamt fachlich und budgettechnisch eingehalten werden.
Zwei kleinere Anpassungen wurden in Absprache mit bzw. auf Anregung vom Projektleiter umgesetzt, auf dem Projekttreffen kommuniziert und durch eine geänderte TVB am 02.05.2023 dem Projektträger übermittelt:

- AP 3.4) „Verifikation der Optimierungsergebnisse mit Hilfe der Fertigung eines Demonstrators und Vergleich der thermischen Eigenschaften mit einem Standardkühlkörperprofil“ [2 PM]

  - Nach Rücksprache mit den assoziierten Partnern für die Kühlkörperherstellung hat sich 2022 herauskristallisiert, dass die Schnittstellen zu Kühlkörper-Herstellern nicht realisierbar sind, da die tatsächlichen Kühlkörpergeometrien von untergeordneten Schnittstellen der Zulieferer abhängen, die man nicht nutzen kann.

    ⇨ Falls eine Schnittstelle zu Kühlkörper-Herstellern erfolgt, wird diese nicht direkt an das Optimierungsframework angebunden, sondern in die Simulationsumgebung von PE-Systems integriert. Das Fraunhofer IISB nutzte stattdessen die Schnittstellen für die PE-Systems Komponenten und baute darauf basierend Use-Cases für diese LE-User-Group auf.

- AP 6) „Optimierung des Designs von Kondensatoren“ [3 PM]

  - Es stellte sich 2022 heraus, dass dies nicht realisierbar ist, da das Kondensator-Design nicht über die geplante digitale Schnittstelle, sondern „manuell“ erfolgen wird.

    ⇨ stattdessen wurden die 3 PM für ein AP 3.5 genutzt:
    Evaluation verfügbarer Ansätze zur Visualisierung großer, hochdimensionaler Datensätze (1.5PM) /beschleunigter Schaltungssimulation durch parallelisierte SPICE Berechnungen (1.5 PM).

2. Fachliche Inhalte und Ergebnisse

Zu Projektstart wurde eine knapp dreißigseitige Studie „State of the Art Optimization Methods“ erstellt, die aus einer mathematischen, algorithmischen Sicht die aktuellen Optimierungsansätze sowie deren Vor- und Nachteile darlegte. Mit dieser Studie wurde die Basis gelegt, die im ersten Schritt in die Implementierung der genetischen Algorithmen (SPEA2 und NSGA2) und langfristig in die Neuentwicklung der „Continuously Adapting Random Sampling“ (CARS) Methode mündeten.

Der Schwerpunkt lag bei Projektbeginn auf der Konzeptionierung eines Optimierungsframeworks, zur effizienten Abarbeitung von über einer Million zeitlich stark variierenden Simulationsjobs. Das in Python entwickelte Framework, genannt „PyFlow“ (Abb. 1), ermöglicht die parallele Abarbeitung von Simulationsjobs und bietet (im Gegensatz zu damals verfügbaren Open-Source-Lösungen wie RAY) den Vorteil der direkten Integrierbarkeit von Optimierungsansätzen.

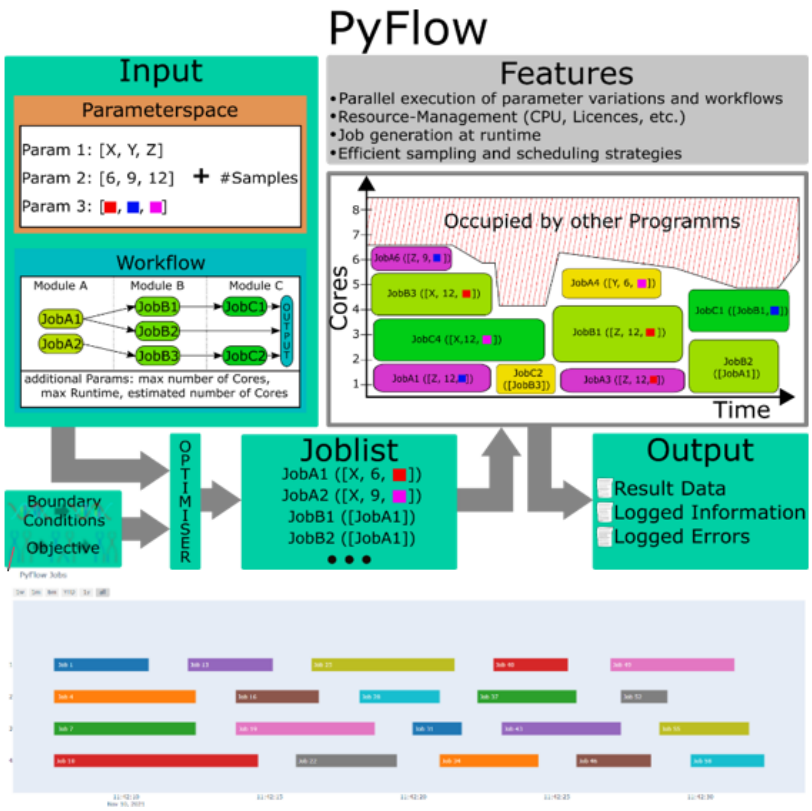

Abb. 1: Übersicht und Aufbau des Pyflow Frameworks (oben), sowie Nachweis der parallelen Abarbeitung auf vier Kernen (unten).

Die wichtigsten Merkmale der PyFlow Implementierung:

- Implementierung eines parallelen Workflows für zeitlich variable Simulationen in Python
- Automatisierte Ausführung und Auswertung von Simulationsaufträgen
- Ressourcenverwaltung: Neue Aufträge starten, sobald vorherige Aufträge abgeschlossen wurden und Ressourcen/Ergebnisse verfügbar sind
- Offene Schnittstellen zu Optimierungsansätzen und Inklusion von genetischen Algorithmen (SPEA2 und NSGA2) sowie deren Kaskadierung
- Robuste Implementierung, Ressourcenmanagement und Doku
- Konfiguration mittels Text-Dateien im YAML-Format

Die Integration externer Anwendungen wie den etablierten Schaltungssimulatoren LTSpice, NgSpice und Simba sowie eine allgemeine Python -Schnittstelle sowie die Verwaltung von Simulationsaufträgen über standardisierte, anwendungs-unabhängige Konfigurationsdateien, wurden iterativ und unter Berücksichtigung des Nutzerfeedbacks (durch die Projektpartner) umgesetzt.
In verschiedenen Tests wurde die Beschleunigung der Schaltungs-Auswertungen mittels PyFlow auf parallelen Systemen gezeigt, insbesondere im Vergleich zu LTspice-internen Ansätzen (Abb. 2).
Darüber hinaus wurden Optimierungsstrategien für leistungselektronische Schaltungen implementiert, bei denen mehrere Optimierungsansätze mit unterschiedlichen „Exploration vs. Exploitation“ Strategien kombiniert wurden (sog. Kaskadierung).

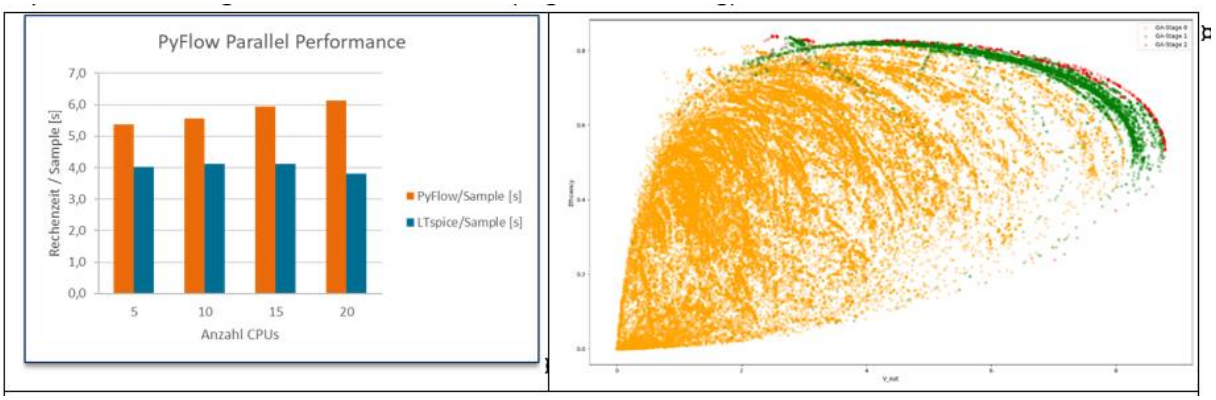

Abb. 2: Beschleunigung der Schaltungs-Auswertungen mittels PyFlow (links) und kaskadierte GAs für die Pareto-Optimierung von Effizienz und Ausgangsspannung eines Boost-Konverters:
Stage 1: orange -> SPEA2 und NSGA2 (65k Samples), Stage 2: grün -> NSGA2 (6k Samples) und Stage 3: rot -> NSGA2 (50 Samples).

Bei der Kaskadierung von SPEA2 und NSGA2 hat sich die Wahl der Parameter für Mutation und Rekombination als entscheidend herausgestellt, um die Stärken beider Algorithmen optimal auszuschöpfen. In SPEA2 kann eine höhere Mutationsrate zu Beginn der Suche eingesetzt werden, um die Vielfalt zu erhöhen und neue Lösungsräume zu erkunden. NSGA2 kann von einer geringeren Mutationsrate profitieren, wenn es darum geht, die gefundene Pareto-Front zu verfeinern und die Exploitation zu verstärken.
Die Wahl einer unterschiedlichen Rekombinationsstrategie, wie z.B. Simulated Binary Crossover (SBX) in NSGA2 und Uniform Crossover in SPEA2, kann helfen, die Durchmischung der Lösungen zu verbessern. Durch die Anpassung der Parameter für Mutation und Rekombination in den verschiedenen Phasen der Kaskadierung, konnte der Algorithmus dynamisch zwischen Exploration und Exploitation wechseln, und lokale Optima, die im Parameterraum derartiger LE-Schaltungen vorkommen überwinden und die globale Pareto-Front effektiver erreichen.
Durch die Anpassung der Mutationsrate in der Kaskadierung gelang es, in der Anfangsphase mehr zufällige Sprünge zu machen und in späteren Phasen gezieltere Verbesserungen zu erzielen. Die unterschiedlichen Rekombinationsmethoden in SPEA2 und NSGA2 konnten die genetische Diversität der Population bewahren und gleichzeitig die Konvergenzrate erhöhen.
Durch viele Tests konnte gezeigt werden, dass der Mehrwert der Kaskadierung mit variablen Mutations- und Rekombinationsparametern in der verbesserten Anpassungsfähigkeit und Robustheit des Algorithmus liegt, was zu einer höheren Ergebnisqualität bei unterschiedlichen Schaltungstopologien führt.
Um die Nutzung bei zunehmender Zahl an Simulationsauswertungen zu erleichtern, wurde mit der Erweiterung „OptiFlow“ ein zusätzliches Framework mit direkten Schnittstellen zur Jobverwaltung „PyFlow“ bereitgestellt sowie eine Übersicht der Best-Practice Parameter hinterlegt. Darauf aufbauend wurden beide Frameworks für hoch-parallele Simulationsausführungen, mit Azure Cloud Ressourcen, weiterentwickelt und mit den Projektpartnern getestet.
Basierend auf der Rückmeldung der Ingenieure und Anwender investierte man viel Zeit in eine interaktive und iterative Visualisierung der großen Datenmengen. Es wurde angestrebt, dass Nutzer die Datensätze aus mehreren hunderttausend multi-dimensionalen Simulationen effizient erkunden und selbstständig filtern können.

II Eingehende Darstellung CODAPE

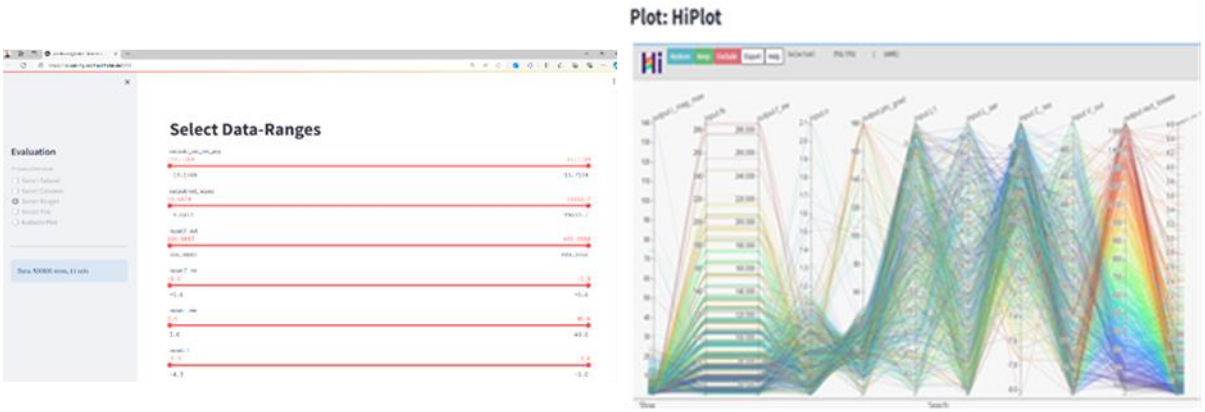

Abb. 3: Import einer CSV Lösungsdatei mit 500.000 samples und der Vorauswahl der Parameterbereiche (links) und die darauf angepasst, interaktiv nutzbare Visualisierung (rechts) basierend auf einer Kombination von Streamlit und HiPlot.

Final wurde eine Kombination von Streamlit und HiPlot implementiert und die jeweiligen offenen Frameworks mit anwendungsspezifischen Schnittstellen und Erweiterungen angepasst.
Es wurde damit möglich, bis zu einer Million 20-dimensionale Datenpunkte rein browserbasiert ohne große Latenz darzustellen.
Darüber hinaus wurden der sog. PHATE-Ansatz eingebunden und getestet. Dieser Ansatz bietet mehrere Vorteile für die Analyse großer Mengen mehrdimensionaler Daten aus Schaltungsauswertungen. PHATE kann nichtlineare Beziehungen zwischen Datenpunkten effektiv erfassen, was bei komplexen Schaltungen mit nicht-trivialen Interaktionen zwischen Komponenten besonders nützlich ist. Die Fähigkeit, sowohl lokale als auch globale Strukturen zu bewahren, ermöglicht es Ingenieuren, sowohl detaillierte Zusammenhänge zwischen ähnlichen Schaltungszuständen als auch übergreifende Muster im gesamten Datensatz zu erkennen. Die Reduzierung der Dimensionalität auf eine niedrigdimensionale Darstellung, erleichtert die Visualisierung und Interpretation großer Datenmengen, was bei der Analyse komplexer Schaltungen von großem Wert sein kann. Die Robustheit gegenüber Rauschen, macht PHATE besonders geeignet für reale Schaltungsdaten, die oft mit Messungenauigkeiten und Störungen behaftet sind. Zudem ermöglicht die Vielseitigkeit von PHATE, verschiedene Aspekte der Schaltungsanalyse - von Simulationsdaten bis hin zu Messwerten - mit einem einzigen Werkzeug zu untersuchen, was die Konsistenz und Vergleichbarkeit der Analysen verbessert.
Parallel zu den Fraunhofer IISB Arbeiten wurde – insbesondere von Block und PE-Systems – ein großer Fokus auf die Beschleunigung der Simulationsauswertungen gesteckt. Insbesondere approximative Schaltungsauswertungen (ohne detaillierte Berücksichtigung von Nichtlinearitäten) zeigten hier, das Potential in weniger als einer Sekunde, erste Tendenzen des Systemverhaltens gut aufzulösen. Die damit gesteigerte Anzahl an Auswertungen pro Zeit konnte zwar gut mit der beschriebenen Visualisierungslösung dargestellt werden, jedoch konnten bei derartigen Datenmengen die genetischen Algorithmen ihren Vorteil nicht ausspielen und wurden sogar von Monte-Carlo (MC) Ansätzen (zufällige Parameterwahl) übertroffen.
Ein genetischer Algorithmus muss nicht nur Lösungen bewerten, sondern auch zusätzliche Schritte wie Selektion, Kreuzung (Crossover) und Mutation durchführen. Obwohl diese Schritte im Vergleich zur Auswertung möglicherweise ebenfalls wenig Zeit in Anspruch nehmen, summieren sie sich im Vergleich zu einem reinen Random Search (MC), der nur Lösungen generiert und bewertet. Wenn die Schaltungsauswertung sehr schnell ist und zusätzlich noch auf vielen Kernen parallel läuft, können diese zusätzlichen Schritte des GAs im Verhältnis zur eigentlichen Evaluationszeit einen erheblichen

Rechenaufwand darstellen, sodass der Vorteil des genetischen Algorithmus gegenüber Random Search verschwindet.
Basierend auf diesen Erkenntnissen wurde die „Continuously Adapting Random Sampling“ (CARS) Methode entwickelt, implementiert und anwendungsspezifisch optimiert. Die CARS-Methode basiert auf einer Reinforcement Learning inspirierten Präferenz-Metrik, die „gute“ Parameter-Bereiche häufiger untersucht und „schlechte“ zunehmend ignoriert. Als Grundlage dient eine vektorisiert berechenbare Softmax-Funktion, die es ermöglicht, bis zu einer Million Parameter-Vorschläge innerhalb weniger Sekunden zu generieren. Diese Methode erlaubt eine effiziente und schnelle Anpassung der Parameter, was besonders bei Simulationen mit kurzen Simulationszeiten von Vorteil ist. Die Softmax-Funktion ist eine mathematische Funktion, die oft in der Wahrscheinlichkeitstheorie und im maschinellen Lernen verwendet wird. Sie wandelt eine Liste von Werten in eine Wahrscheinlichkeitsverteilung um deren Summe gleich 1 ist. Dies ermöglicht es, die Wahrscheinlichkeit zu berechnen, dass ein bestimmter Parameter-Vorschlag ausgewählt wird, basierend auf seiner relativen Güte im Vergleich zu anderen Vorschlägen. In den etablierten Python-Frameworks sind diese Methoden sehr effizient implementiert und können deswegen hier auch für große Datenmengen äußerst effizient in den Gesamtalgorithmus eingebaut werden.

Zusätzlich zu diesen grundlegenden Konzepten wurden mehrere Erweiterungen und Heuristiken implementiert, um die Effizienz der CARS-Methode weiter zu steigern. Diese Erweiterungen ermöglichen es, Simulationsergebnisse auf benachbarte Parameterbereiche zu übertragen und wenig vielversprechende Simulationen zu vermeiden. Dies bedeutet, dass die Ergebnisse einer Simulation nicht nur für den spezifischen Parameter-Satz gelten, sondern auch für ähnliche Parameter-Sätze in der Nähe. Diese Methode basiert auf der Annahme, dass benachbarte Parameterbereiche ähnliche Eigenschaften aufweisen und daher ähnliche Simulationsergebnisse erwartet werden können.
Durch diese Erweiterung konnte die Anzahl der notwendigen Simulationen reduziert und die Gesamteffizienz des Optimierungsprozesses erhöht werden.

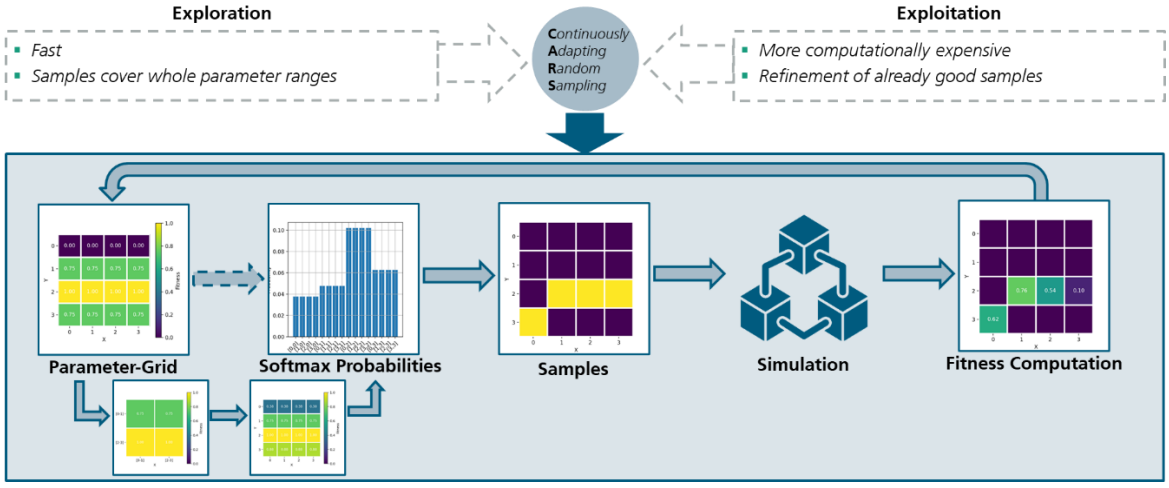

Abb. 4: Übersicht und Ablauf der CARS-Methode

Weiterhin wurden Heuristiken ergänzt, um die CARS-Methode basierend auf einer Gesamtanzahl an Simulationen bzw. einem definierten Zeitbudget zu konfigurieren. Diese Heuristiken ermöglichen eine dynamische Anpassung, der Methode an die spezifischen Anforderungen der jeweiligen Simulationsumgebung und tragen zur weiteren Optimierung des Optimierungsworkflows bei. Die

Heuristik wurde entlang vieler Untersuchungen an LE-Schaltungen entwickelt und ist optimiert für resonante Topologien, da Entwickler durch deren starke Nichtlinearität vor große Herausforderungen gestellt wurden.
Um diese Methode im Projekt iterativ zu verbessern, wurde eine einheitliche Konfigurations- und Kommunikationsschnittstelle definiert, mit der die Optimierungsanfragen verarbeitet werden können. Diese Schnittstellen dienen als Grundlage für die Integration und Kommunikation mit dem PE-Systems Interface in AP 2.
Erste Tests der CARS-Methode und der implementierten genetischen Algorithmen wurden für drei exemplarische Leistungselektronik-Anwendungen durchgeführt: einen Boost Converter und zwei LLC-Resonanzwandler. Auf Grund der hohen Dimensionalität der Lösungen (z.B. 6 Inputs und 4 relevante Outputs) stellte die Aufbereitung und Analyse der Daten eine große Herausforderung dar. Insgesamt zeigt sich, dass die CARS-Methode in schnellen Simulationssetups, mit einem festen Zeitbudget für die Optimierung, signifikante Vorteile gegenüber traditionellen Ansätzen bietet. In Abb. 5 wird eine LLC-Schaltung für resonante Wechselrichter mit vier Arbeitspunkten hinsichtlich Effizienz optimiert (unter Berücksichtigung von Randbedingungen für die Ausgangsleistung und Phasenwinkel) und es zeigt sich nach einem festen Zeitbudget von 60min, dass CARS mit über vierzigtausend gültigen Lösungskandidaten die etablierten Ansätze (Random und GA) um drei Größenordnungen übertrifft.
Die CARS-Methode hat sich als vielversprechende Lösung für die Optimierung von leistungselektronischen Systemen in schnellen Simulationsumgebungen erwiesen. Durch die Kombination von aus dem Reinforcement Learning inspirierten Techniken, effizienten Sampling-Methoden und flexiblen Konfigurationsmöglichkeiten bietet sie eine hohe Effizienz und Anpassungsfähigkeit.
Die erfolgreichen Tests an realen Anwendungsbeispielen unterstreichen das Potenzial dieser Methode für die zukünftige Entwicklung und Optimierung von leistungselektronischen Systemen.

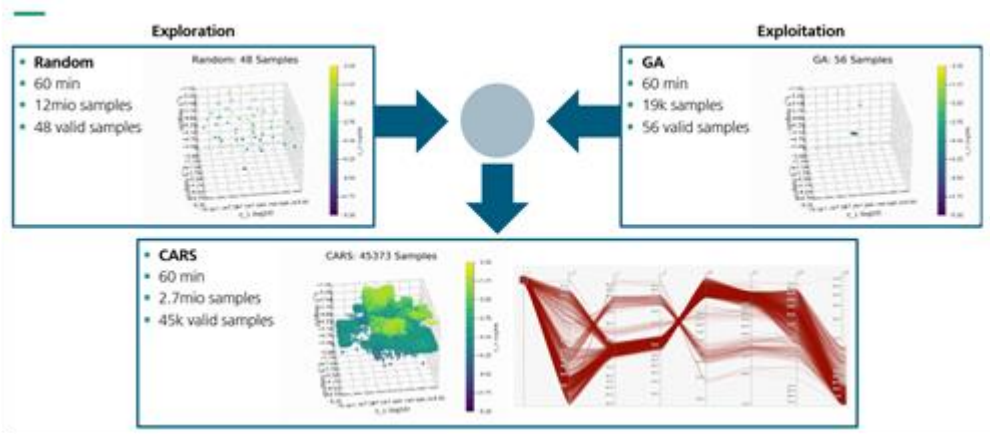

Abb. 5: LLC-Schaltungs-Optimierung hinsichtlich Effizienz mit drei Ansätzen (Random, GA, CARS) für ein festes Zeitbudget von 60 Minuten

Im Folgenden soll die Integration des Optimierungsframeworks CARS mit der Simulationsumgebung von PE-Systems detaillierter beleuchtet werden. Ziel ist, eine effiziente, nahtlose und sichere Interaktion zwischen den beiden Systemen zu gewährleisten.
Zunächst wurden in enger Zusammenarbeit mit PE-Systems die notwendigen Schnittstellen sowohl zur Benutzeroberfläche als auch zur Simulationsumgebung detailliert definiert (AP 2.1). Diese Definitionen bildeten die Grundlage für die anschließende Implementierung und den Test der Schnittstellen (Appendix A).

Die Schnittstelle zur Simulationsumgebung wurde erfolgreich in Betrieb genommen und für den aktuell verfügbaren Funktionsumfang umfassend getestet. Über diese Schnittstelle wird ein Dictionary, dass die erforderlichen Parameterwerte enthält, an die Simulationsumgebung von PE-Systems übermittelt. Diese Parameter dienen als Eingabewerte für die Schaltungssimulation. Nach der Durchführung der Simulation liefert die Simulationsumgebung ein Dictionary mit den relevanten Simulationsgrößen zurück. Dieser bi-direktionale Datenaustausch ermöglicht es dem Optimierungsframework, präzise Simulationsergebnisse zu erhalten, die für die weitere Optimierung genutzt werden können.
Parallel dazu wurde die Schnittstelle zur Benutzeroberfläche entwickelt, um die Anwendungen der User-Group abzubilden. Diese Schnittstelle ermöglicht es, Optimierungs-Settings über eine API zu übermitteln. Das übermittelte Dictionary enthält unter anderem variable Parameter, Nebenbedingungen und Zielfunktionen, die für den Optimierungsprozess entscheidend sind. Nach Abschluss der Optimierung werden die untersuchten Parameter zusammen mit den jeweiligen Simulationsergebnissen an die Benutzeroberfläche zurückgeliefert. Dieser Prozess stellt sicher, dass die Benutzer stets Zugriff auf die aktuellen Optimierungsergebnisse haben und diese in ihre weiteren Entscheidungen einfließen lassen können.
Die Implementierung dieser Schnittstellen ist in Abb. 6 dargestellt und ermöglicht eine reibungslose Interaktion zwischen dem Optimierungsframework und der Schaltungssimulationsumgebung. Dies führt zu einer erheblichen Verbesserung der Effizienz und Genauigkeit der Optimierungsprozesse, da die Simulationsergebnisse direkt in den Optimierungszyklus einfließen und somit eine kontinuierliche Verbesserung der Schaltungsdesigns gewährleistet wird. Arbeiten zum Projektende und auch jetzt im Nachgang des Projekts zielen darauf ab, die Benutzerfreundlichkeit weiter zu erhöhen und sicherzustellen, dass alle erforderlichen Anpassungen vorgenommen werden, um den spezifischen Anforderungen der User-Group gerecht zu werden.

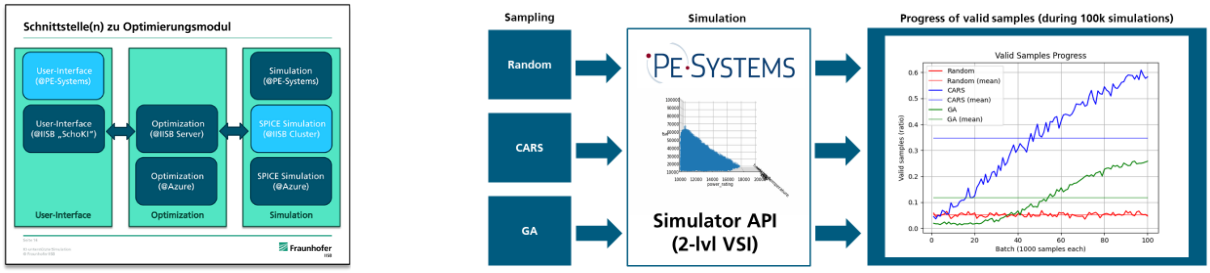

Abb. 6: Schnittstellen der Optimierungsmodule (links), hellblau mit manuell und dunkelbau mit automatischer Interaktion, sowie die Validierung für ein Testsetup von PE-Systems und hunderttausend Simulationen (rechts).

3. Der voraussichtliche Nutzen, insbesondere die Verwertbarkeit des Ergebnisses - auch konkrete Planungen für die nähere Zukunft - im Sinne des fortgeschriebenen Verwertungsplans

Parallel zur Entwicklung des Optimierungsframeworks für die Interaktion mit PE-Systems wurde eine separate Softwarelösung mit Schnittstellen zu LTSpice entwickelt. Diese Lösung umfasst das CARS/CODAPE-Optimierungsframework, das erfolgreich auf dem IISB-Server bereitgestellt wurde. Ein zentrales Element dieser Lösung ist das „SchoKI“ Web-Interface (**Sch**altungs**o**ptimierung mit **KI**), das den Nutzern ermöglicht, LTSpice-Netzlisten hochzuladen und deren Optimierung zu konfigurieren. Dabei können Parameter, Zielfunktionen und Nebenbedingungen festgelegt werden, um den Optimierungsprozess individuell anzupassen.

Ein herausragendes Merkmal dieser Softwarelösung ist die automatisierte LTSpice-Simulation, die auf den vom Optimierungsframework generierten Sample-Kandidaten basiert. Diese Automatisierung ermöglicht eine effiziente Durchführung von Simulationen, ohne dass manuelle Eingriffe erforderlich sind. Die Ergebnisse der Optimierung und Simulation werden anschaulich visualisiert, unter anderem durch die Verwendung von Parallelkoordinatenplots mittels HiPlot. Diese Visualisierungsmethode erleichtert es den Nutzern, komplexe Zusammenhänge und Trends in den Daten zu erkennen und fundierte Entscheidungen zu treffen.
Insgesamt zeigt die Umsetzung (siehe Abb. 7), dass die entwickelte Softwarelösung eine leistungsfähige und vielseitige Plattform für die Optimierung und Simulation von Schaltungen darstellt. Die bisherigen Ergebnisse sind vielversprechend und bieten eine solide Grundlage für zukünftige Erweiterungen und Anpassungen an spezifische Nutzerbedürfnisse. Es ist perspektivisch geplant, Kunden die Nutzung dieser Softwarelösung als Software-as-a-Service gegen Entgelt anzubieten.

Abb. 7: Workflow des IISB-Workflows „SchoKI“ zur Optimierung von LT Spice Schaltungen.

4. Während der Durchführung des Vorhabens dem Zuwendungsempfänger bekannt gewordenen Fortschritt auf dem Gebiet des Vorhabens bei anderen Stellen

Nachfolgend werden, unterteilt in die verschiedenen Kalenderjahre, die Fortschritte bei anderen Stellen für die Bereiche Workflows, Schaltungssimulatoren und Optimierung dargestellt und anschließend aus einer Retrospektive ganzheitlich eingeordnet.

2021:

Im Bereich der Parallelisierung von Python Prozessen ist mit „RAY“ (https://docs.ray.io) ein neuartiger Ansatz, zur einfachen Adressierung verteilter Hardware-Ressourcen, von einer Forschungsgruppe der Berkeley University, veröffentlicht worden.
Weiterhin wurde mit „CUSPICE“ (http://ngspice.sourceforge.net/cuspice.html) eine bisher relativ unbekannte SPICE Implementierung mit GPU-Unterstützung (aus 2014) entdeckt, die im Hinblick

auf AP 3.2 eine ausgiebigere Nutzung von GPU-Ressourcen ermöglichen kann und weiter untersucht werden soll.
Abschließend wurde von Google Ende 2020 ein Paper im Artverwandten Bereich der Chip-Design-Optimierung veröffentlicht (https://www.nature.com/articles/s41586-021-03544-w.epdf), welches einerseits die Vorteile und Potentiale von automatisierten Design Prozessen aufzeigt und andererseits neue Ansätze wie die Nutzung von Reinforcement Learning für Optimierungsprobleme liefert.

2022:

SIMBA (https://simba.io/) ist ein digitales Tool zur Leistungselektronik-Simulation. Dabei werden ähnliche Funktionalitäten wie in dem geplanten Design-Demonstrator zur Verfügung gestellt. Unter anderem z.B. die Spezifikation von Schaltungen in einem Online-Portal und die integrierte Schaltungssimulation. Allerdings wird kein integriertes Optimierungsframework zur Verfügung gestellt, was den Mehrwert des CODAPE Designdemonstrators und insbesondere des AP3 hervorhebt.
Aufbauend auf einer internen Masterarbeit zur Nutzung von Reinforcement Learning für Optimierungsfragestellungen, wurden entsprechende Veröffentlichungen im Leistungselektronik Bereich weiterverfolgt. Den Mehrwert dieses Ansatzes zeigen erste Veröffentlichungen, mit Schwerpunkten auf der Anwendung, z.B. [1] oder auch auf der Reinforcement Learning Methodik, z.B. [2].

2023:

QSPICE (QSPICE™ Simulator - Qorvo) ist ein Schaltungssimulator für leistungselektronische Systeme der im Q1/2023 veröffentlicht und vom Entwickler von LTSpice implementiert wurde. Er wurde in der Community sehr positiv aufgenommen, da viele anwendungsspezifische Neuerungen eingebaut wurden. Wir nutzten die ersten Releases für eine Evaluation wie im Jahr 2022 für das Programm SIMBA (siehe Report 2022).
Ähnlich wie LTspice und Ngspice verwendet QSPICE adaptive Zeitschritte, also Ansätze in denen bei (Schalt-)Ereignissen die Zeitschrittweite verringert wird, was zu vielen Auswertungen führt, wenn sich die transienten Signale schneller ändern als die vorgegebenen Zeitschritte.
Erste Vergleiche zeigen, dass die Rechenzeiten dieses neuen Programms für die von uns betrachteten Konvertertopologien schneller sind als für die Vorgängersoftware LTSpice (Speed-Up um Faktor 2-3), jedoch SIMBA weiterhin um einen Faktor 5-10 schneller ist (wenn das passende Domain-Know-How in der Wahl der Zeitschritte eingebracht werden kann).

Während der Projektlaufzeit hat der Themenbereich „digitale Simulationsworkflows“ Fahrt aufgenommen und insbesondere das Framework „Ray“ hat eine fulminante Entwicklung genommen. Ray wurde von zwei Doktoranden des RISELab der UC Berkeley, Philipp Moritz und Robert Nishihara, entwickelt. Die Forscher wollten einen Rahmen schaffen, der viele der verschiedenen Elemente kombiniert, die für die Ausführung von Anwendungen für maschinelles Lernen oder Deep Learning in der realen Welt erforderlich sind. Mittlerweile wird es auch für das Training von OpenAI Modellen z.B. ChatGPT verwendet und hat über 22.000 commits und über 33.000 stars auf Github. Dank der frühzeitigen Berücksichtigung dieses Frameworks konnten unsere Schnittstellen im Code passend

designed werden und insbesondere die Entwicklung Richtung CARS (dessen Funktionalität so in Ray nicht verfügbar ist) vorangetrieben werden.
Bei den Schaltungssimulatoren handelt es sich um sehr spezielle Software, die sowohl eine sehr niedrige Einstiegsschwelle für Ingenieure bieten als auch exakte und schnelle Lösungen liefern muss. Wir sind zu der Erkenntnis gekommen (siehe IISB-Vortrag zum Abschlusstreffen - GPT. 6), dass für unsere Fragestellungen der Optimierung, mit wenig Variationen in der Topologie, alle am Markt existierenden Lösungen noch großes Verbesserungspotential bieten; konkret sehen wir Speed-Ups von mehreren Größenordnungen durch GPU-Implementierungen, die idealweise in einem Nachfolgeprojekt umgesetzt werden sollen.
Die Nutzung von Reinforcement Learning Strategien zur Optimierung war ein großer Erfolg – uns ist bis heute keine ähnlich effiziente Implementierung bekannt. Insbesondere ermöglicht sie eine – der Erwartungshaltung der Kunden (Leistungselektronik-Ingenieure, Schaltungsdesigner, etc.) entsprechende – Bereitstellung von guten Lösungskandidaten, die mit zunehmender Rechenzeit verbessert wird.

5. Die erfolgten oder geplanten Veröffentlichungen des Ergebnisses nach Nr. 5 der NKBF/NABF

- **Kruse, G., Happel, D., Ditze, S., Ehrlich, S., & Rosskopf, A. (2022, October). Parameter optimization of llc-converter with multiple operation points using reinforcement learning.** In *2022 IEEE 20th Biennial Conference on Electromagnetic Field Computation (CEFC)* (pp. 1-2). IEEE.
  Die Optimierung elektrischer Schaltungen ist ein komplexer und zeitaufwändiger Prozess, der zunehmend auch von fortschrittlichen Algorithmen durchgeführt wird. In dieser Arbeit wird ein Reinforcement-Learning-Ansatz zur Optimierung eines LLC-Wandlers an mehreren Betriebspunkten vorgestellt, wobei der trainierte RL-Agent in der Lage ist, neue Optimierungsprobleme in weniger als 50 Schritten mit Wirkungsgraden über 90 % zu lösen und somit das Potenzial bietet, Experten-basierte Design-Prozesse in der Leistungselektronik durch datengetriebene Strategien zu ergänzen.
- **Ditze, S., Ehrlich, S., Happel, D., & Rosskopf, A. (2023, March). Asymmetric resonant tank design for a bi-directional CLLC resonant converter in G2V and V2G operation.** In *2023 IEEE Applied Power Electronics Conference and Exposition (APEC)* (pp. 1358-1365). IEEE.
  In dieser Arbeit wird eine Entwurfsmethode für einen asymmetrischen CLLC-Resonanzwandler vorgestellt, die auf dem zeitlichen Verhalten des Wandlers basiert. Ein Algorithmus durchsucht eine Simulationsdatenbank nach passenden Resonanzkreiskonfigurationen für verschiedene Betriebszustände, und die Methode wird durch Messungen an einem bi-direktionalen 3,6 kW-Ladegerät validiert.
- **Happel, D., Brendel, P., Rosskopf, A., & Ditze, S. (2023). Continuously Adapting Random Sampling (CARS) for Power Electronics Parameter Design**. *arXiv preprint arXiv:2310.10425*.
  Bisher wurde Design-Optimierung in der Leistungselektronik meist durch detaillierte Optimierungsansätze oder grobe Raster-Suchverfahren mit schnellen Simulationen angegangen. Die neue Methode "Continuously Adapting Random Sampling" (CARS) bietet einen kontinuierlichen Ansatz dazwischen, der schnelle und umfangreiche Simulationen ermöglicht und sich dabei zunehmend auf vielversprechende Parameterbereiche konzentriert; ihre Leistung wurde in drei Anwendungsfällen der Leistungselektronik evaluiert und zeigt konkurrenzfähige Ergebnisse im Vergleich zu genetischen Algorithmen.

5. Appendix

```yaml
##################################################################################################################
# general settings
# key -> (str) key to authenticate at API
# ID -> (str) unique name (used as ID for asynchronous process)
# sampling_name -> (str) name appended to results folder (should be short and informative)
# samples -> (list of int) number of samples (upper bound of samples for each workflow
#             e.g. [10, 20, 30] for workflows [Random, RBF, LTspice])
#             also allows to specify a one-element list, which will be applied for all workflows
# method -> (str) specification of pre-defined workflows [cars_spice, pyflow_spice, pyflow_spice_cost, ...]
#           see (_controller)/api_controller/config/* for pre-defined workflows
##################################################################################################################

key: VuRy[redacted]
ID: xyz
sampling_name: test
samples: [5000]
method: pyflow_spice


##################################################################################################################
# Module settings
# overwrite ANY key provided in module configs (/*module*/config/defaults.yaml)
# API config overwrites defaults recursively (setting individual elements of nested dictionaries is also possible)
#
# *module-name* e.g. "cars" -> first-level key aka module name
#   *config-key* e.g. "strategy" -> second-level key aka module attribute
##################################################################################################################

cars:
  strategy: balanced

numpy_fitness:
  auto_normalization: True

standalone_pyflow:
  sampling:
    method: GA


##################################################################################################################
# Parameters
# name -> (str) parameter name (key to be queried from PE-Systems API, column in DataFrame)
# type -> (str) [linear, log, grid, categorical] (grid should be used for discrete numerical values, e.g. [1, 2, ...]
#                                                 categorical should be used for non-numerical values, e.g. [A, B, ...])
# vals -> (list of lists of floats) [[min, max]] or [[value11, value12, ...], [value21, value22, ...], ...] or [[value]]
# nop -> (int) optional, if provided and > 1, parameter will be independently sampled for multiple operating points
##################################################################################################################
parameter:
  - name: C1
    type: log
    vals: [[1e-9, 1e-3]]

  - name: L1
    type: log
    vals: [[1e-6, 100e-3]]

  - name: fsw
    type: log
    vals: [[100, 1e6]]



##################################################################################################################
# Objectives
# name -> (str) objective name (column name in DataFrame)
# type -> (str) [min, max, target, min-difference]
# weight -> (float) weight for scaling (applied after normalization) default 1.0
# vals -> (list of lists of floats) optional list of target values (required if type==target)
##################################################################################################################
objectives:

  - name: vmean
    type: target
    weight: 1.0
    vals: [[12]]

  - name: eff_tot
    type: max
    weight: 1.0



##################################################################################################################
# Boundary Conditions
# name -> (str) boundary name (column name in result DataFrame)
# type -> (str) [smaller (<), smaller-equal (<=), larger (>), larger-equal (>=), in-range (>= ... <=)]
# vals -> (list of lists of floats) one list for every operating point, for in-range [min, max] list for each OP
# weight -> (float) weight for penalty if BC is violated (applied after normalization) default 100.0
##################################################################################################################
boundaries:

  - name: vmean
    type: in-range
    vals: [[11.5, 12.5]]
    weight: 100

  - name: vrip
    type: in-range
    vals: [[0, 2]]
    weight: 100
```

Appendix A
Beispiele eines yaml-files zur Kommunikation via der API Schnitstelle