VA PY：使用反向传播进行电路优化

Anthony Yang[1]，Ron Dubinksy[2],Yu-Neng Wang[1],Sara Achour[1]，†斯坦福大学，美国加利福尼亚州斯坦福[2]独立研究员，美国华盛顿州雷德蒙德

†通讯作者{anthyang、wynwyn、sachour}@stanford.edu；rondubinsky1@gmail.com

arXiv:2611.28554v1 [cs.PL] 21 Nov 2026

摘要——**模拟电路的参数灵敏度使基于梯度的设计优化方法得以应用，并拓展了多种灵敏度分析的可能性。然而，对于使用工艺设计套件的电路，要计算精确的设计参数灵敏度仍然是一项挑战，因为这些工艺设计套件使用的 Verilog-A 器件模型包含数千行代码。通过仿真估计梯度的有限差分方法，根据所采用的方法，对每个梯度需要进行 $N+1$ 至 $2N$ 次仿真；当优化的参数较多时，这一开销可能高得难以承受。我们提出 VAPY，这是一种 Verilog-A 到 JAX 的转译器，可将 Verilog-A 器件模型编译为可自动微分的 JAX 代码，从而支持使用基于 JAX 的电路仿真器 Circulax 进行优化。VAPY 采用污点分析，确保仅使用 JAX 兼容调用重写必要的控制流，从而降低控制流开销。我们使用 VAPY 和 Circulax 优化了在 SkyWater SKY130 工艺设计套件中实现的三个模拟电路，该工艺设计套件基于 BSIM4 器件模型构建。VAPY 生成的模型梯度与有限差分梯度高度一致，在减少超过 $40\times$ 次电路求解的情况下，实现了相当的设计质量；与贝叶斯优化相比，损耗最高降低了 40%。**

索引词—**模拟电路优化、自动微分、紧凑模型**

I. 引言

对于电子设计自动化工具而言，模拟混合信号电路的自动化设计优化仍然是一项挑战。设计优化涉及优化电路参数，例如晶体管宽度，以最大化根据电路仿真结果计算得出的某个品质因数。参数敏感度或梯度信息对于此类问题非常有价值，因为它可用于通过基于梯度的方法优化电路；此外，对于其他使用场景也具有独立价值，例如良率分析，在这类分析中，工程师关注品质因数对特定设计参数的敏感度。

目前，由于模型本身复杂，针对由 PDK 提供的晶体管模型构成的电路，计算设计参数灵敏度颇具挑战。电路原生的解析方法（例如电路伴随法）虽然可用于计算参数灵敏度，但仅支持简单的（例如，1 级）晶体管模型，且通常需要人工操作 [1]–[5]。有限差分方法（FD）通过扰动各个参数来近似计算梯度，能够处理复杂模型，但根据具体方法，每次计算梯度都需要进行 $N+1$ 次或 $2N$ 次电路仿真。因此，对于涉及大量参数的优化问题，FD 方法的扩展性较差。

本研究着重利用自动微分（AD）计算参数灵敏度；这项技术通常用于从机器学习模型中解析推导梯度。我们提出了 VAPY，这是一种转译器，可将 PDK 提供的器件模型编译为 Python 代码，然后利用 AD 通过单次电路仿真直接计算参数灵敏度。

借助 VAPY，我们能够为 SkyWater SKY130 工艺设计套件自动生成基于 Python 的器件模型 [6], ，并对采用 SKY130 实现的电路相对于其设计参数进行自动微分。VAPY 能够对 SKY130 中使用的 BSIM4 器件模型进行自动微分，而 Xyce（一款旗舰级 C++ 仿真器，支持对采用 Verilog-A 语言语法子集描述的模型进行自动微分 [7]）目前尚不支持这一功能。

使用 VAPY 推导的灵敏度函数，每次计算梯度只需进行一次仿真。随着参数数量增加，与 FD 相比，这大幅减少了计算每个梯度所需的仿真次数。借助 VAPY，我们能够使用梯度下降方法优化 SKY130 电路，同时精确计算参数灵敏度，并找出 VAPY 灵敏度比 FD 更准确的情况。

II. 背景

模拟电路由参数化器件的网表描述，例如晶体管、电容器、电阻器和开关。其拓扑结构定义了制约器件行为的物理约束。这些网表通常以 SPICE 电路的形式指定，并使用多种电路仿真器进行仿真，包括 SPECTRE、ngSpice 和 Xyce。通常，器件物理模型内置于仿真器中。器件实例提供设计参数，而其余参数则由制造该器件的晶圆代工厂指定。工艺设计套件（PDK）是晶圆代工厂提供的库，其中的器件模型经过参数化，以反映使用其工艺制造的器件的行为。器件模型

在工艺设计套件中，参数化方式可能有所不同，以对 p 型和 n 型晶体管、阈值电压、工艺角以及允许的工作条件进行建模。这些晶圆代工厂参数也可能取决于器件设计参数，例如宽度和长度。

近期进展。直到不久以前，参数化器件的物理模型还都是用 C++在仿真器中实现的。源设备接口（OSDI）由 SemiMod GmbH 于 2022年推出，Open

源设备接口（OSDI）使得兼容的仿真器能够加载独立编译的器件模型。包括 ngSpice 在内的多个知名仿真器采用了这一标准。2023年，SemiMod 发布了 OpenVAF 编译器，它可以通过 LLVM 中间表示将 Verilog-A 器件模型编译为二进制文件。Verilog-A 是一种模拟行为建模语言，许多 BSIM 晶体管模型系列都以此语言实现。这些开源模拟电子设计自动化工具领域的最新进展为 VAPY 的开发提供了条件。

自动微分和机器学习。 机器学习库（例如 PyTorch 和 JAX）使用自动微分（AD），通过对程序的基本操作应用链式法则来计算精确导数，从而在瞬时工作点获得准确的非近似梯度，并消除手动处理过程。

最近，这些框架已被应用于电路仿真器中。VAJAX [8] 和 Circulax [9] 是基于 JAX 的电路仿真框架，分别于 2026年和 2025年发布 [10]。VAJAX 专为模拟电路仿真的 GPU 加速而构建，其自动微分支持仅限于器件残差和雅可比矩阵，而不支持任意参数梯度。

III. VAPY 转译器

VAPY 转译器通过源到源转换（§III-A）将 VA 器件模型转换为语义等价的 Python 抽象语法树（AST），随后将其编译为与 Circulax 兼容的、支持自动微分的 JAX 实现。Circulax 是一个基于 Python 的 SPICE 电路仿真器（§ IV）。IV-A 节介绍了如何将 VAPY 生成的器件模型与 Circulax 结合，通过反向传播执行电路优化。

A. Verilog-A 到 *Python* 转译器

VAPY 的转译前端会解析 VA 程序，并将其转换为 Python AST。VAPY 的解析器和中间表示扩展自 OpenVAF 模型编译器；该编译器可将 Verilog-A 器件模型转换为符合 OSDI 标准的二进制代码。

OSDI 标准。 VAPY 根据开放源代码器件接口（OSDI）标准，将 VA 器件模型编译为 Python 表示。OSDI 标准定义了一种应用程序二进制接口，可确保与常见的开源 SPICE 仿真器互操作。VAPY 扩展了 OpenVAF-reloaded [11]，这是一个基于 OpenVAF 的开源 Verilog-A 编译器，可生成符合 OSDI v0.4 的模型。这个分支改进了面向 VA 的 OpenVAF 编译器，其目标版本为 OSDI v0.3。

转译。 VAPY 要求器件模型能够由 OpenVAF 编译，并将其表示为受支持的标量方程。VAPY 会将代码转译为一种 Python 表示，其中包含setup、init和eval 函数，其形式类似于符合 OSDI 标准的模型中的setup model、setup instance和 eval 。在 VAPY 的抽象中，setup 函数会根据工艺设计套件参数 ϕ创建一个新的器件模型对象 m 。 $\vec{x}$ =init(m,θ) 函数会实例化一个器件实例，并使用实例特定的参数 θ（例如宽度和长度），然后返回一个能够捕获运行时参数的固定大小向量 $\vec{x}$ 。$\vec{r}_i$,$\vec{r}_q$ =eval($\vec{x}$,$\vec{v}$) 函数接收器件实例向量 $\vec{x}$和固定大小的电压向量 $\vec{v}$，该电压向量包含节点电压和支路电压，并返回固定大小的电抗向量 $\vec{r}_q$ 和固定大小的电阻向量 $\vec{r}_i$。对于每个由工艺设计套件实例化的唯一器件，都会调用一次setup 函数；每个器件实例调用一次init 函数；而eval函数则会根据仿真情况，为每个实例调用多次。

B. 将 *Python* 转换为可追踪的 *JAX* 函数

VAPY 的 JAX 转译器遍会将生成的 Python 代码转换为 JAX Python 代码。JAX 支持数值计算代码的自动微分和即时编译，从而实现程序的 GPU 加速和并行化，并支持使用基于梯度的方法进行优化。JAX 要求所有依赖数据的控制流（例如条件语句）都必须使用等效的 JAX 控制流（例如调用 jax.lax.cond ）实现，并禁止原地修改数组。函数中可追踪的变量越多，自动微分生成的代码和 JAX 编译就越复杂，尤其是在被追踪的代码包含许多jax.lax.cond 调用时。

追踪优化。 为降低生成的 JAX 代码的复杂度，VAPY 的 JAX 转译器遍采用了一种称为污点分析的程序分析技术。该转译器接收污点变量（TV）列表，该列表指定了用户希望对其进行自动微分的 PDK、器件实例和电压输入参数的子集。随后，污点分析通过在函数抽象语法树中迭代传播变量依赖关系，以保守方式识别依赖于 TV 的变量和条件分支。要使 BSIM4 模型支持 AD，这些已识别的运算必须可追踪。所有不依赖于 TV 的变量都会解析为运行时间常量。

条件语句。 使用污点分析技术识别出的依赖于 TV 的条件语句会转换为 JAX 控制流，而其他所有条件语句则保留为 Python 条件语句。这样既能保留预期的梯度计算路径，又能降低不必要的复杂度。

逻辑。标量数学运算会转换为jax.numpy运算，以支持追踪。布尔类型转换会变为(x!= 0) ，而浮点类型转换会被移除，以避免在没有定义值的情况下计算追踪器，因为缺少此类值会引发错误。整数字面量会转换为浮点数字面量，以防止int64 溢出。离散运算中使用的整数会保留。具有未定义行为的数学运算（平方根、对数、除法）会替换为安全数学运算；对于未定义的输出，这些安全数学运算会返回有限替代值。这是因为在模型的向量化仿真中，jax.lax.cond 调用的两个分支都会被计算，未使用的分支可能会在这些运算符中触发未定义行为。

IV. CIRCULAX 集成与器件库

VAPY 转译器生成的器件模型与 Circulax 兼容。Circulax 是一款基于 JAX 构建、类似 SPICE 的模拟器，支持模拟电路的交流、直流和瞬态分析。第

IV-A 节介绍 VAPY 生成的 Circulax 器件模型，IV-B 节介绍 VAPY 生成的工艺设计套件库。

A. VAPy 生成的可微器件模型

Circulax 要求将每个器件模型定义为一个函数，该函数接受输入电压 $\vec{v}$ 和运行时参数 $\vec{x}$ 作为输入。VAPY 使用 JAX 的自动微分程序追踪函数逻辑，并推导出一个灵敏度函数，用于计算反应输出和电阻输出相对于 $\vec{x}$ 和 $\vec{v}$的灵敏度。

当用户提供类似 SPICE 的电路描述时，Circulax 会组合这些器件模型，生成一个电路求值函数，并在仿真过程中调用该函数以生成仿真结果 $\vec{o}$。随后使用反向传播计算 $\partial\vec{o}/\partial x$ 和 $\partial\vec{o}/\partial\vec{v}$。VAPY 结合链式法则以及运行时参数相对于 θ 的灵敏度来计算 $\partial\vec{o}/\partial\vec{\theta}$:

$$\partial\vec{o}/\partial\vec{\theta} = \partial\vec{o}/\partial\vec{x} \cdot \partial\vec{x}/\partial\vec{\theta} \tag{1}$$

运行时参数灵敏度 $\partial\vec{x}/\partial\theta$ 通过追踪 init 函数并对其执行自动微分来计算。为 Circulax 生成的 init(ϕ,θ) 函数结合了转译器输出中的init和 setup函数逻辑，并且只能相对于 θ进行追踪。借助 VAPY 的链式法则扩展，我们可以计算仿真结果相对于器件参数的灵敏度。

兼容性优化。 JAX 支持将程序编译为 XLA；XLA 是 JAX 的默认编译后端，可用于加速。电路求值函数会被降级为 XLA。作为一项兼容性优化，init 函数会进行自动微分，但不会降级为 XLA，因为其中包含 XLA 不支持的运算符。由于 JAX 不支持追踪由主机执行的 Python 与 XLA 实现构成的组合，因此会分别计算init和eval的灵敏度，再通过（1）将其组合。

性能优化。Circulax 在单次批处理操作中并行求解器件。我们修改了 Circulax，使其可以选择按顺序求解器件物理，以减少反向传播期间的内存占用。

形状变换。 JAX 转译器会将eval 函数的输出重塑为 Circulax 所需的格式。

B. PDK 支持

VAPY 会根据可微器件模型和 *PDK* 规范（一个 JSON 文件）自动生成 PDK 库。该文件将晶圆代工厂定义的器件参数 ϕ 描述为晶体管宽度和长度的函数。PDK 规范会从 PDK 卡中自动提取。VAPY 会构建一个 PDK 库，将每个 PDK 器件模型名称映射到专用的器件映射。专用器件映射会将分档区间（定义参数适用的宽度和长度区间）转换为部分求值后的器件初始化函数版本，并填入 ϕ。在每次优化器迭代中，都会根据每个器件当前的宽度和长度，为其选择合适的 初始化函数。

表 I：器件、工艺设计套件和电路信息

BSIM 4 器件模型				
语言	LoC	# 参数		
		总计	离散	连续
Verilog-A	12654	899	33	866
Python	46165	899	33	866
JAX	58900	899	33	866

SKY130 工艺设计套件 ● 所有晶体管模型均使用 *BSIM4* 器件模型				
晶体管模型	Used	# 参数		
		固定	Free 离散	Free 连续
低阈值电压 1.8V NMOS 晶体管	✓	885	2	9+3†
1.8V NMOS 晶体管	✓	885	2	9+3†
5.0V 漏极／10.5V 栅极 NMOS 晶体管		885	2	9+3†
低阈值电压 1.8V PMOS 晶体管		885	2	9+3†
1.8V PMOS 晶体管	✓	885	2	9+3†
高阈值电压 1.8V PMOS 晶体管		885	2	9+3†
pfet_g5v0d10v5		885	2	9+3†

基准电路 ● 所有电路均使用 *SKY130*					
电路	# 设计参数	# 晶体管-istors	Res-晶体管	Caps	开关
跟踪保持电路	1	1	0	1	0
自举	9	1	0	3	5
LDO	36	6	3	2	0

† 非设计参数（§V-A）。

C. 局限性

VAPY 的转译器不支持 for 和 while 循环。对于每种器件模型，所有循环都有界，因此我们手动展开了每个循环。循环展开是一种标准的编译器优化，因此很容易实现自动化。

V. 评估

A. 基准电路

表 I 展示了基准电路，这些电路均采用 SKY130 工艺设计套件实现。表中说明了需要优化的设计参数数量以及各类器件的数量。本文评估了三个基准电路——跟踪保持电路、自举采样开关（自举）电路以及低压差稳压器（LDO）。所有设计均采用工艺设计套件提供的晶体管模型，以及无源器件和开关的理想模型。SKY130 工艺设计套件是一个采用 54 级 BSIM4 器件模型的开源工艺设计套件。它向用户开放了十二个连续设计参数，其中三个是工作温度（我们不对其进行优化），还有漏极和源极扩散区中的平方单位数，这些参数被硬编码为零。底层 BSIM4 模型包含 12.7K 行 VA 代码，并开放了 33 个离散参数和 866 个连续参数，其中除 14 个外，其余参数均由晶圆代工厂设定。

跟踪保持电路。 跟踪保持电路是一种模拟电路，通常用于在测量电压时将其保持恒定，以供模数转换器使用。该电路取自[12] ，并在我们的分析中用作合理性测试。在跟踪模式下，输出电压跟随输入电压。在保持模式下，输出保持在跟踪模式下最后采样到的电压。工作模式由晶体管上的时钟控制的栅极电压设定，该电压控制晶体管是否导通，使输入电压能够通过其漏源端子。该电路的品质因数（FoM）

旨在最小化电路瞬态电压轨迹相对于参考轨迹的均方根误差（RMSE），同时最小化晶体管的宽度。我们对晶体管的宽度进行优化，初始宽度为 4µm。

自举。自举采样开关用于使 MOSFET 的栅源电压保持近似恒定，不受输入信号影响，以尽可能减小采样失真。针对一个晶体管优化九个参数。该基准电路取自 [13]。此电路的品质因数旨在尽可能减小晶体管面积，同时将三次谐波失真（HD3）保持在 -75 分贝以下。为此任务优化全部九个晶体管设计参数。这些品质因数指标根据瞬态分析计算得出。指定的 SKY130 器件最小晶体管长度 L = 0.15µm 用作初始值。初始宽度 W =60.75µm 根据解析推导出的 405 宽长比计算得出，该宽长比可使 HD3 低于 -75 分贝。

低压差稳压器。低压差稳压器用于在输入电压或负载电流变化时维持输出电压恒定。该电路基于 [14]的 SimpleLDO 电路。针对四组晶体管优化三十六个参数。此电路的品质因数综合优化目标为：尽可能减小晶体管面积和直流工作点偏置，尽可能提高电源抑制比（即尽可能减小电源抑制比不足），并尽可能降低静态功耗。电源抑制比是一项交流指标，用于衡量对输入信号中电源纹波的抑制能力。这些指标根据直流工作点分析和交流分析计算得出。由于低压差稳压器通常在不同负载和频率下工作，品质因数会综合考虑多种负载电流大小和输入信号频率。

B. 评估方法

每个电路的参数都通过一个能够反映该电路品质因数的损失函数进行优化。对于特定电路的所有优化运行，均使用相同的初始参数配置。对于每种优化器，我们将最大迭代次数设为 40 次，但 BO-4X 除外，其迭代次数为 160 次。所有优化运行均在配备 16 GB 内存和 Intel Core Ultra 7 155H CPU 的 x86-64 CPU 机器上执行。为提高性能，我们对所有优化运行中的电路仿真进行即时编译。

基于梯度的优化。为对电路设计参数进行基于梯度的优化，我们使用 Adam 优化器，学习率固定为 0.05，其余参数均采用默认值。参数更新中的缩放项用于处理参数值的不同量级。

评估中使用了两种计算梯度的方法。我们使用通过对自动微分的 VAPY 生成的模型进行反向传播所计算的梯度（ADiff）。作为基准，我们还使用有限差分方法（FDiff）来近似梯度。由于二阶中心有限差分能够提供良好的梯度近似，因此采用该方法；在参数边界处则退回使用一阶前向/后向有限差分。

表 II：跟踪保持电路宽度参数与品质因数

方法	W (μm)	# 求解次数	Loss	RMSE
BO-S	4.879	41	$2.786 \cdot 10^{-7}$	$5.278 \cdot 10^{-4}$
BO-4X	4.879	161	$2.786 \cdot 10^{-7}$	$5.278 \cdot 10^{-4}$
FDiff	4.430	42	$3.862 \cdot 10^{-6}$	$1.965 \cdot 10^{-3}$
ADiff	4.430	14	$3.826 \cdot 10^{-6}$	$1.965 \cdot 10^{-3}$

贝叶斯优化。我们使用黑盒贝叶斯优化（BO-S 和BO-4X）基准评估基于梯度的优化。贝叶斯优化是一种无梯度优化方法，使用高斯过程代理模型近似参数与损失函数之间的关系。BO-S 执行与基于梯度的方法一样，进行40次优化迭代；BO-4X 执行则使用 $4 \times 40 = 160$ 次迭代。BO-4X 基准分配了额外的求解器迭代次数，以适应更高维的搜索空间，从而为贝叶斯优化提供比基于梯度的方法更大的迭代预算来寻找解决方案。

指标。 对于每种优化方法，我们报告所发现的最优损耗以及构成品质因数的电路特定指标。为了定量评估 VAPY 模型生成的参数灵敏度相对于有限差分基线的准确性，我们报告 ADiff 与FDiff 梯度之间的最大峰值归一化误差（MPNE），该指标反映两者之间的最高误差。

为评估性能，我们以兆字节（MB）为单位报告优化和编译的内存占用。优化运行时间细分为每次电路求解所需的时间（求解时间）、每次优化器迭代所需的时间（每次迭代时间）以及总优化时间（总计）。我们还报告每个优化器的电路求解次数（$\#iter \times \#sims/iter$）。

C. 优化器结果分析

表 II、表 III 和表 IV 报告了各个优化器在每个基准电路上的参数化结果。随着设计参数数量增加，基于梯度的方法找到的参数化结果优于BO-S和BO-4X。VAPY 生成的ADiff流程以远少于FDiff 的电路求解次数，得到几乎相当的参数化结果。对于跟踪保持电路（表 II），ADiff 比 FDiff 少进行 66.7% 的电路求解，虽然损耗较差，但宽度小于BO-S和 BO-4X。对于自举电路（表 III），ADiff仅用 FDiff 9.3% 的电路求解次数就找到了最优解；与BO-S 和BO-4X相比，它找到的参数化结果损耗更低、面积更小，且 HD3 达标。对于低压差稳压器（表 IV），ADiff仅用FDiff2.4% 的电路求解次数就找到了最优解，同时损耗降低了 40%，电源抑制比不足更低，直流工作点偏置也更低。总的来说，随着搜索空间扩大，基于贝叶斯优化的方法更难找到良好的参数化结果。

D. 灵敏度分析

接下来，我们比较ADiff和FDiff梯度的一致性。我们发现，ADiff 和FDiff梯度在各

表 III：自举电路参数与品质因数

方法	W (μm)	L (μm)	ad	as	pd	ps	sb	sa	sd
BO-S	60.75	0.15	0	0	0	0	0	0	0
BO-4X	49.751	0.15	0	0	0	0	0	0	0
FDiff	44.744	0.15	0	0	0	0	0	0	0
ADiff	44.743	0.15	0	0	0	0	0	0	0

方法	# 求解	loss	晶体管面积（μm^2）	谐波失真（分贝）
BO-S	41	0.152	9.113	-81.598
BO-4X	161	0.124	7.463	-79.149
FDiff	441	0.112	6.712	-77.757
ADiff	41	0.112	6.711	-77.757

表 IV：低压差稳压器

方法	# 求解	loss	晶体管 area (μm^2)	max 静态功率 (mW)	PSRR 短-fall (dB)	直流工作点。点 Bias (mV)
BO-S	41	5.163	976.818	0.0813	3.198	10.694
BO-4X	161	5.163	976.818	0.0813	3.198	10.694
FDiff	1677	3.060	1028.918	0.0838	2.070	10.342
ADiff	41	3.076	1034.709	0.0840	2.066	10.361

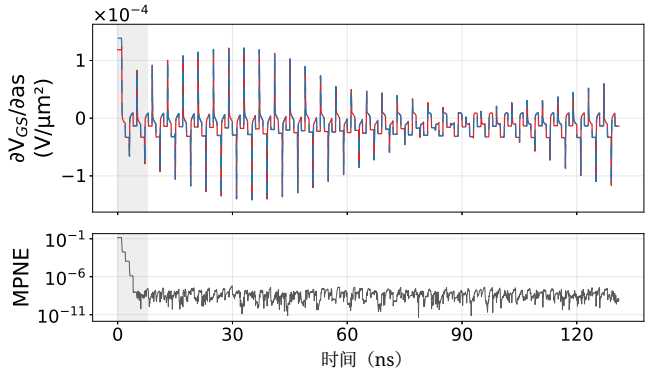
个基准电路中均一致，验证了其在瞬态、交流和直流分析中的准确性。对于跟踪保持电路，在瞬态分析期间，所有参数 p 的输出电压梯度 $\partial V_{out}/\partial p$的 MPNE 均低于$10^{-4}$ 。对于自举电路，在瞬态仿真期间，所有参数的 HD3梯度 $\partial HD3/\partial p$和 V_{GS} 梯度 $\partial V_{GS}/\partial p$ 的 MPNE 分别低于 10^{-3} 和 0.15。对于低压差稳压器，在交流分析期间，电源抑制比梯度$\partial PSRR/\partial p$ 的 MPNE 低于 10^{-2} ；所有参数的电压梯度 $\partial V_{out}/\partial p$的 MPNE 均低于 10^{-5} 。总而言之，ADiff和 FDiff之间的梯度差异相当小，即使在最差情况下也是如此。

图 1a 和 1b 定性展示了 $\partial V_{GS}/\partial as$ 和 $\partial PSRR/\partial L_{pair}$的梯度与 MPNE。每个图的上方显示 ADiff和FDiff 计算出的梯度，下方显示 MPNE。堆叠图共用 x 轴。总体而言，各轨迹在定性上相符。我们注意到，在自举 V_{GS} 图中，ADiff 和FDiff 起初并不一致——在此例中，FDiff 报告的梯度有误，因为它对 $t = 0$ 处初始直流电压的差分非常小，以至于直流误差会显著影响其梯度。相较之下，ADiff 采用瞬时导数。定量分析中 $\partial V_{GS}/\partial p$的 MPNE 反映了这一差异。排除这段初始时间区间后，MPNE 低于 10^{-3}。

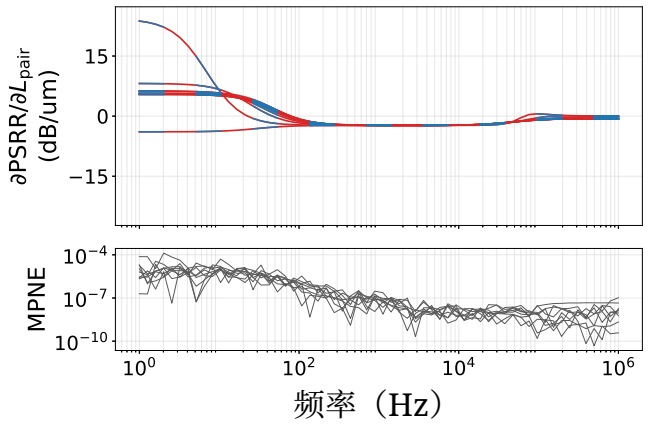
E. 性能分析。

表 V 展示了各优化器的运行时间和内存占用细分情况。在编译时，ADiff 由于需要执行自动微分遍历以直接计算梯度，因此比基线方法需要更多内存。在优化过程中，与.....相比，ADiff 每次电路求解所需的时间更长

由于反向遍历，其运行时间比基准方法更长。不过，与FDiff相比，它每次迭代所需的电路求解次数更少，因此单次迭代时间和总运行时间更接近FDiff ，具体取决于基准测试——



（a） $\partial V_{gs}/\partial as$ 自举电路



（b） $\partial PSRR/\partial L_{pair}$ 低压差稳压器

图 1：自举和低压差稳压器的参数灵敏度。 ■是ADiff， ■ 是FDiff， ■是 MPNE。

尤其是在进行瞬态仿真时。对于进行交流分析的情况（例如低压差稳压器），由于交流系统的反向遍历中会计算混合二阶导数，ADiff的求解时间明显高于未采用自动微分的基准方法，而仅前向遍历的计算成本极低。

我们观察到，尽管ADiff 需要更多内存和更长运行时间，但随着设计参数数量的增加，这些性能指标并不会大幅上升。在相同的 2,000 步瞬态设置下，对比跟踪保持电路（$N = 1$）和自举（$N = 9$）可见，尽管参数数量增加了 $9\times$ ，ADiff的每次求解运行时间和优化内存分别仅增加 $2.2\times$和 $1.3\times$。相比之下，FDiff的迭代时间增加了 8 倍。

造成这种性能差异的原因有几个。值得注意的是，实现反向遍历的 Circulax 问世约一年，主要使用 JAX 的通用优化与仿真库（例如 optimistix和 diffrax）来构建电路仿真算法。因此，Circulax 几乎没有利用电路仿真算法的任何领域特定特性。VAPY 也尚未应用

表 V：性能分析

方法	跟踪保持电路（瞬态（*.TRAN*）分析，*2000* 个时间步）				
	内存（*MB*）		运行时间（s）		
	优化	编译	求解时间	每次迭代时间	总计
BO-S	52	703	12.08	12.38	495.24
BO-4X	60	645	13.21	13.29	2126.94
FDiff	22	711	12.59	37.77	528.85
ADiff	153 5555 159.92 159.92 2238.84				
方法	自举（瞬态（*.TRAN*）分析，*2000* 个时间步）				
	内存（*MB*）		运行时间（秒）		
	优化	编译	求解时间	每次迭代时间	总计
BO-S	25	743	25.58	26.22	1048.6
BO-4X	65	683	31.08	31.27	5003.91
FDiff	8	732	27.00	297.63	11905.30
ADiff	193	5576	355.98	364.88	14595.37
方法	低压差稳压器（直流（*.OP*）和交流小信号（*.AC*）分析）				
	内存（*MB*）		运行时间（秒）		
	优化	编译	求解时间	每次迭代时间	总计
BO-S	22	338	8.40	8.60	344.35
BO-4X	231	333	16.74	16.84	2695.19
FDiff	35	329	0.13	5.35	214.01
ADiff	201	5014	34.92	35.80	1431.91

性能优化来转换模型，而是依赖 JAX 发掘优化机会。随着这两个库的进一步开发，编译和求解时间有望缩短。

VI. 相关工作

计算电路梯度的方法。模拟设计自动化中的“伴随灵敏度分析”技术通过构建辅助伴随电路来计算灵敏度，此后已得到广泛研究 [1]–[4]。这些研究专注于器件级优化，针对延迟或增益等电路特定指标进行优化，需要对电路元件进行线性化，且通常仅适用于简单的晶体管模型。相比之下，VAPY 能够通过自动微分对采用复杂得多的晶体管模型的电路进行梯度计算；我们的实验涉及基于 BSIM 第 54级晶体管模型的晶体管模型。有限差分方法也用于计算梯度，而无需通过电路模型进行自动微分。对于 N 参数梯度，前向-后向有限差分方法需要$N+1$ 次仿真，但会导致更大的梯度误差；中心差分方法需要 $2N$ 次仿真，梯度误差则更小。VAPY 通过对仿真中的数值计算进行反向传播，能够从单次电路仿真中直接计算梯度。

基于梯度的电路优化工具。

Xyce 和 Circulax 等电路仿真器提供了对器件模型和电路仿真进行自动微分的基本支持 [7],[9]。Xyce 支持对器件模型进行直接灵敏度分析和伴随灵敏度分析，但仅支持对手动求导的线性器件以及由其 ADMS 后端生成的模型使用解析导数。其他模型，例如 BSIM4 和 BSIM-SOI，则退回使用有限差分方法，这种方法的梯度误差较大，且仿真开销高 [7]。此外，对于一阶以上的导数，例如与交流仿真的自动微分相关的导数，Xyce 也会退回使用数值近似。相比之下，VAPY 的转译器能够编译器件模型并将其直接转换为兼容 JAX 的 Python 代码，从而支持对模型进行端到端自动微分，以计算高阶参数灵敏度，并实现 GPU 加速。

此外，Xyce 模型没有 Python 表示，而 VAPY（我们的转译器）基于通用 AD，VAPY 编译的模型可直接与 Python 优化和学习框架结合。

WiCkeD 等生产级电路优化器使用有限差分方法来近似梯度，并结合多种算法优化技术 [15]。原则上，VAPY 的微分方法可以替代上述有限差分技术。

模拟电路优化对于晶体管尺寸设计，先进的模拟电路优化通常将仿真器视为无梯度黑盒，并采用贝叶斯优化 [16]，强化学习 [17], [18],或语言模型 [19]。近期将贝叶斯优化与电路参数梯度相结合的研究提高了采样效率 [5]。然而，该框架使用伴随灵敏度分析，因此仅适用于由简单晶体管模型构成的电路。VAPY 通过为复杂器件模型（例如 BSIM4）提供精确梯度，补充了这种优化方法，从而能够对更复杂的电路进行混合优化。

VII. 结论

我们介绍 VAPY，这是一种 VA 到 JAX 转译器，可对工艺设计套件中常用的紧凑器件模型进行自动微分，从而能够在 Python 仿真和机器学习生态系统中对模拟电路进行端到端的基于梯度的优化。由于 VAPY 可通过追踪直接从器件模型推导出参数灵敏度函数，因此只需进行一次电路仿真，即可计算精确的设计参数灵敏度，从而避免有限差分近似带来的 $O(N)$ 仿真成本。我们证明，VAPY 生成的模型能够成功集成到 Circulax 中，并以远少于有限差分方法所需的电路求解次数来优化 SKY130 电路。此外，我们还证明，VAPY 支持的基于梯度的优化能够在初始点周围的多维空间中找到比贝叶斯优化更好的最优解，展现出高效混合方法的潜力。通过进一步的算法和性能优化，我们相信还能进一步降低绝对运行时间和内存占用。

参考文献

[1] S. Director 和 R. Rohrer，“广义伴随网络与网络灵敏度”，*IEEE* 电路理论汇刊，第 16 卷，第 3 期，第 318–323 页，1969 年。[2] C. Visweswariah、R. A. Haring 和 A. R. Conn，“电路优化中的噪声考量”，*IEEE* 集成电路与系统计算机辅助设计汇刊，第 19 卷，第 6 期，第 679–690 页，2000 年。[3] W. Hu、Z. Ye 和 Y. Wang，“针对与多个时间点相关的目标函数进行伴随瞬态灵敏度分析”，载于*2020*年第*57*届 *ACM/IEEE* 设计自动化会议（*DAC*）。IEEE，2020 年，第 1–6 页。[4] J. Li、D. Ahsanullah、Z. Gao 和 R. Rohrer，“时域伴随灵敏度的电路理论”，*IEEE* 集成电路与系统计算机辅助设计汇刊，2023 年。[5] R. Lyu、A. Zhao、Y. Meng、K. Zhu、Z. Bi、C. Yan、F. Yang、D. Zhou 和 X. Zeng，“重新审视基于灵敏度的模拟电路尺寸设计：采用具备导数感知能力的贝叶斯优化和误差抑制伴随分析”，载于第 *43* 届 *IEEE/ACM* 计算机辅助设计国际会议论文集，2024 年，第 1–9 页。[6] 谷歌和 SkyWater Technology，“SkyWater 开源工艺设计套件”，https://github.com/google/skywater-pdk，2020 年。[7] E. Keiter、J. Verley、T. Russo、R. Hoekstra、H. Thornquist、T. Takhtaganov、R. Schiek、T. Mei、R. Pawlowski、K. Santarelli、T. Coffey、S. Hutchinson、P. Sholander、K. Aadithya、G. Hennigan、L. Musson、T. Kolda、R. Hooper、M. Hsieh、C. Warrender、A. Meek、S. Verzi、B. Bond、A. Lundin、B. Van Bloeman Waanders、E. Rankin、H. Watts、D. Fixel、B. Fett、N. Johnson、L. Boucheron Spence、D. Barnes、D. Day、L. Waters、P. Campbell、D. Baur 和 D. Shirley，“Xyce(TM) 并行电子电路模拟器 v.7.6”，[计算机软件] https://doi.org/10.11578/dc.20230414.1，2013 年 10 月。 [在线]。可访问：https://doi.org/

10 11578 20230414 1 。/dc. 。ChipFlow，“VAJAX：支持 [8] OpenVAF Verilog-A 的基于 JAX 的 SPICE 模拟器”，https://github.com/ChipFlow/vajax，2026年。[9] gdsfactory，“Circulax：基于 JAX 的可微电路仿真”，https://github.com/gdsfactory/circulax，2026年。[10] J. Bradbury、R. Frostig、P. Hawkins、M. J. Johnson、C. Leary、D. Maclaurin、G. Necula、A. Paszke、J. VanderPlas、S. Wanderman-Milne 和 Q. Zhang，“JAX：Python+NumPy 程序的可组合变换”，2018 年。 [在线]。可访问：http://github.com/jax-ml/jax [11] A. Buermen，“GitHub 上的 OpenVAF-reloaded 源代码”，2025年。 [在线]。可访问：https://github.com/arpadbuermen/OpenVAF [12] Boris Murmann，“Ngspice-on-colab”，https://github.com/bmurmann/Ngspice-on-Colab/tree/main，2024年。[13] B. Murmann，“作业 #1”，ICONS 2026：高级集成电路设计大师班系列，夏威夷大学马诺阿分校，2026年。[14] CODA-Team，“AnalogGym”，https://github.com/CODA-Team/AnalogGym/tree/main，2025年。[15] ´A. B˝urmen 和 H. Habal，“通过网格自适应直接搜索计算模拟集成电路的最坏情况性能和良率”，*Informacije MIDEM*，第 45 卷，第 2 期，第 160–170 页，2015 年。[16] W. Lyu、F. Yang、C. Yan、D. Zhou 和 X. Zeng，“通过多目标采集函数集成进行批量贝叶斯优化，以实现自动化模拟电路设计”，载于国际机器学习会议。PMLR，2018 年，第 3306–3314 页。[17] H. Wang、K. Wang、J. Yang、L. Shen、N. Sun、H.-S. Lee 和 S. Han，“GCN-RL 电路设计器：结合图神经网络和强化学习的可迁移晶体管尺寸设计”，载于*2020*年第*57*届 *ACM/IEEE* 设计自动化会议（*DAC*）。IEEE，2020 年，第 1–6 页。[18] K. Settaluri、A. Haj-Ali、Q. Huang、K. Hakhamaneshi 和 B. Nikolic，“Autockt：模拟电路设计的深度强化学习”，载于*2020*年欧洲设计、自动化与测试会议暨展览（*2020*）。IEEE，2020 年，第 490–495 页。[19] Y. Yin、Y. Wang、B. Xu 和 P. Li，“ADO-LLM：基于大语言模型上下文学习的模拟设计贝叶斯优化”，载于第 *43* 届 *IEEE/ACM* 计算机辅助设计国际会议论文集，2024 年，第 1–9 页。