MECH 579 Final Project

Multi-Objective PDE-Constrained Optimization of a CPU Cooling System

Lucas Bessai, 261050265
Aidan Kimberley, 261004905
Department of Mechanical Engineering, McGill University
817 Sherbrooke Street West, Montreal, QC, H3A 0C3

December 17, 2025

1 Introduction

Modern CPUs generate significant heat during operation, especially in high-performance computing applications. To prevent overheating and ensure reliable performance, the processor package must be coupled with an effective cooling system. In the simplified model studied in this project, a fan cools the top surface of a CPU via convective heat transfer, while internal circuit components generate heat within the CPU volume.

The volumetric heat generation is modelled as a linear function of the spatial coordinates on the CPU surface,

$$\dot{E}(x, y) = ax + by + c \quad [\mathrm{W/m^3}], \tag{1}$$

where a, b, and c are abstract parameters summarizing how the circuit components are distributed over the CPU area. Higher local values of $\dot{E}$ correspond to higher component density and therefore higher local heat generation. The design variables (a, b, c) effectively control the slope and offset of this heat source. Figure 1

Convective cooling is driven by a fan blowing air across the CPU surface. The fan velocity v is also treated as a design variable and drives the convective heat transfer coefficient. The temperature field on the CPU evolves according to a transient heat equation and, after sufficient time, reaches a steady-state temperature distribution. The primary physical concern is the maximum steady-state temperature: if this becomes too high, the CPU may throttle or fail.

This project formulates the cooling system design as a multi-objective, PDE-constrained optimization problem. The two objectives are:

- minimize the maximum steady-state temperature on the CPU surface; and
- maximize the fan efficiency, which penalizes unnecessarily high fan speeds.

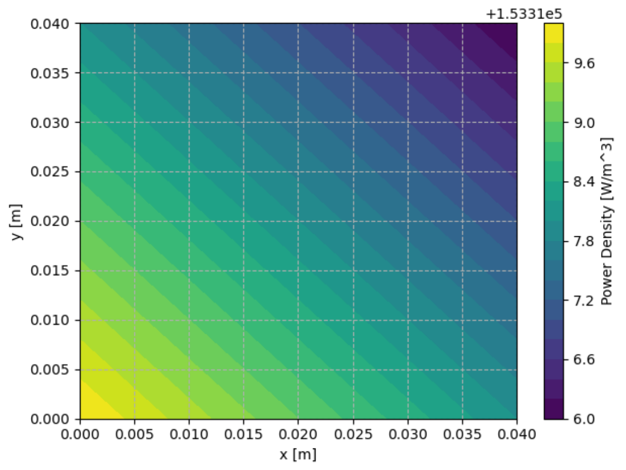


Figure 1: Contours of the power density (heat-generation) field $\dot{E}(x, y) = ax + by + c$ at the optimal parameter values. Function describes a plane in x and y. Regions of high power density correspond to clusters of circuit components.

These objectives are combined using scaling and weights ω_1 and ω_2 to form a single scalar cost. The optimization is subject to (i) an equality constraint on the total CPU power output (fixed at 10 W) and (ii) the steady-state heat equation.

The report first describes the baseline NumPy implementation of the heat equation and the finite-difference (FD) based optimization, and then presents a JAX-based implementation that enables automatic differentiation (AD) through the PDE solver. Finally, sensitivity analysis and Pareto exploration are used to interpret the problem structure and the numerical results.

2 Problem Formulation

2.1 Governing equations

The CPU top surface is represented by a square domain

$$\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \subset \mathbb{R}^2, \quad x \in [0, 0.04]m \ , \quad y \in [0, 0.04]m$$

The temperature field $T(x, y, t)$ satisfies the 2D transient heat equation

$$\rho c \frac{\partial T}{\partial t} - k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) = \dot{E}(x, y; a, b, c), \tag{2}$$

where ρ is the density, c is the specific heat, k is the thermal conductivity, and $\dot{E}(x, y)$ is the volumetric power density.

Convective cooling along the exposed surface is modelled via Robin-type boundary conditions

$$-k \nabla T \cdot \mathbf{n} = h(v,\ T)\,(T - T_\infty) \quad \text{on } \partial\Omega, \tag{3}$$

where T_∞ is the ambient air temperature, $\mathbf{n}$ is the outward normal, and $h(v,\ T)$ is a convective heat transfer coefficient that depends on the fan velocity v. In the provided code, $h(v)$ is a function of the Reynolds and Prandtl numbers consistent with typical correlations between air speed and convective cooling.

The time integration begins from an initial temperature field $T(x, y, 0) = T_0(x, y)$. As $t \to \infty$, the solution approaches a steady-state temperature distribution $T_{\text{ss}}(x, y)$ satisfying

$$-k\left(\frac{\partial^2 T_{\text{ss}}}{\partial x^2} + \frac{\partial^2 T_{\text{ss}}}{\partial y^2}\right) = \dot{E}(x, y; a, b, c), \tag{4}$$

and the boundary conditions. The optimization is carried out on this steady-state solution. The key scalar quantity is the maximum steady-state temperature

$$T_{\max}(v, a, b, c) := \max_{(x,y)\in\Omega} T_{\text{ss}}(x, y). \tag{5}$$

2.2 Fan efficiency model

The fan efficiency is prescribed as a quadratic function of the fan speed v,

$$\eta(v) = -0.02\, v^2 + 0.08\, v, \qquad v \in [0, 40] \text{ m/s}. \tag{6}$$

Efficiency increases with v up to a peak at $v = 20\,\text{m/s}$ and then decreases for higher velocities. Physically, this reflects diminishing returns: higher airflow eventually requires disproportionally more input power for a modest improvement in cooling.

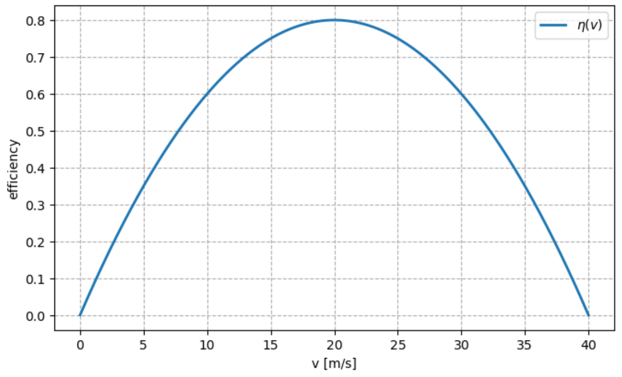


Figure 2: Fan efficiency $\eta(v)$ as a function of fan velocity. Efficiency peaks at $v = 20$ m/s and decreases for higher speeds.

2.3 Constraints and design variables

The total power generated by the CPU is constrained to be 10 W. With a uniform thickness H in the out-of-plane direction and $f(x,y) = ax + by + c$, the total power is

$$P(a,b,c) = H \int_{y_{\min}}^{y_{\max}} \int_{x_{\min}}^{x_{\max}} (ax + by + c)\, dx\, dy. \tag{7}$$

The equality constraint is

$$g(a,b,c) := P(a,b,c) - 10 = 0. \tag{8}$$

The design variable vector is

$$\mathbf{x} = \begin{bmatrix} v & a & b & c \end{bmatrix}^\top. \tag{9}$$

The bounds used in the optimization are set to

$$0.1 \le v \le 30, \qquad -10^8 \le a \le 10^8, \qquad -10^8 \le b \le 10^8, \qquad 0 \le c \le 10^8. \tag{10}$$

The small lower bound on v avoids numerical issues at zero fan speed, while a and b are allowed to be positive or negative to represent different spatial orientations of the linear heat-generation plane.

2.4 Multi-Objective Formulation

The two objectives (minimize maximum temperature, maximize fan efficiency) are combined to form a mulit-objective problem:

$$J(\mathbf{x}) = \omega_1 \frac{T_{\max}(\mathbf{x})}{273\ \text{K}} - \omega_2\, \eta(v), \qquad \omega_1 + \omega_2 = 1, \tag{11}$$

where the maximum temperature is normalized by 273 K so that its scale is comparable to the efficiency, which lies between 0 and 0.8. This normalization prevents one objective from numerically dominating solely due to units.

For the main set of results, the weights are chosen as $\omega_1 = 0.2$ and $\omega_2 = 0.8$, prioritizing fan efficiency while still penalizing high maximum temperature.

The resulting optimization problem is:

$$\begin{aligned} \min_{\mathbf{x}} \quad & J(\mathbf{x}) && (12)\\ \text{s.t.} \quad & g(a,b,c) = 0, && (13)\\ & \mathbf{x}_{\min} \le \mathbf{x} \le \mathbf{x}_{\max}, && (14)\\ & T_{\text{ss}} \text{ satisfies the steady-state heat equation.} && (15) \end{aligned}$$

3 Numpy Implementation and FD-Based Optimization

3.1 Finite-difference discretization and time-stepping

The supplied NumPy code implements a class-based finite-difference solver for the transient heat equation. The spatial domain is discretized into a uniform grid, and the temperature field is stored in a 2D array `self.u`. The explicit time-stepping scheme is

$$T_{i,j}^{n+1} = T_{i,j}^{n} + \frac{\Delta t}{\rho c}\left[k\left(D_{xx}T_{i,j}^{n} + D_{yy}T_{i,j}^{n}\right) + \dot{E}_{i,j}\right], \tag{16}$$

where D_{xx} and D_{yy} are second-order finite-difference approximations of the Laplacian in the x and y directions.

The time step is chosen using a CFL-type condition,

$$\Delta t = \mathrm{CFL}\,\frac{\Delta x\,\Delta y}{\alpha}, \qquad \alpha = \frac{k}{\rho c}, \tag{17}$$

to maintain stability of the explicit scheme. Robin boundary conditions are enforced by modifying the boundary nodes in `self.u` according to (3).

The method `solve_until_steady_state()` advances the solution in time until the ∞-norm of the difference between successive temperature fields falls below a tolerance:

$$\|T^{n+1} - T^{n}\|_{\infty} < 10^{-3}. \tag{18}$$

The resulting T^{n+1} is taken as the steady-state temperature distribution T_{ss}.

3.2 Objective and constraint evaluation

Given a vector of design variables $\mathbf{x} = [v, a, b, c]^{\top}$, the NumPy implementation:

1. sets the fan velocity and updates the convective cooling coefficient;
2. sets the heat-generation parameters a, b, c and updates $\dot{E}(x, y)$;
3. resets the temperature field to the initial condition;
4. calls `solve_until_steady_state()` to obtain T_{ss};
5. computes $T_{\max}$ as the maximum entry of T_{ss};
6. evaluates $\eta(v)$ using (6);
7. constructs $J(\mathbf{x})$ and the constraint $g(a, b, c)$.

The optimization is performed using `scipy.optimize.minimize` with the `trust-constr` method. No analytical Jacobian is supplied, so the solver uses finite-difference (FD) approximations for both the objective gradient and the constraint Jacobian. The constraint is passed as a single equality constraint in the standard dictionary form

```
constraints = [{'type': 'eq', 'fun': constraint_one}]
```

$$g(a, b, c) = 0$$

and the physical bounds on v, a, b, and c are enforced using the `bounds` argument.

```
bounds = [
    (0.1, 30.0),    # v
    (-1e8, 1e8),    # a
    (-1e8, 1e8),    # b
    (0.0, 1e8),     # c
]
```

3.3 FD optimization results

For the baseline NumPy implementation, the initial guess

$$v_0 = 15 \text{ m/s}, \quad a_0 = 10, \quad b_0 = 10, \quad c_0 = 156250,$$

does not satisfy the power constraint, but the optimizer adjusts c during the iterations to enforce $P = 10\,\mathrm{W}$ at the optimum.

`trust-constr` converges in roughly 14 iterations with about 50 function evaluations. The final design variables (for $\omega_1 = 0.2$, $\omega_2 = 0.8$) are approximately

$$\begin{aligned} v^* &\approx 20 \text{ m/s}, \\ a^* &\approx -48.7 \text{ W/m}^4, \\ b^* &\approx -48.7 \text{ W/m}^4, \\ c^* &\approx 1.53 \times 10^5 \text{ W/m}^3, \end{aligned}$$

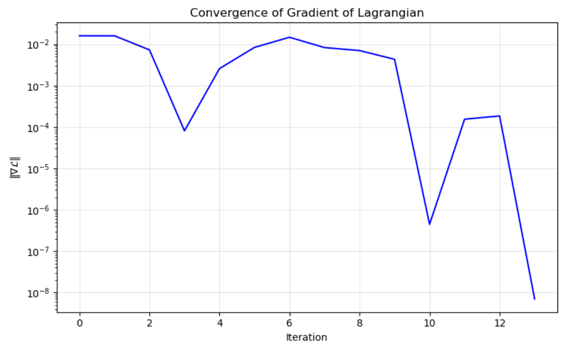
with

$$T_{\max} \approx 321.45 \text{ K } (48.45°\text{C}), \qquad \eta(v^*) \approx 0.80.$$

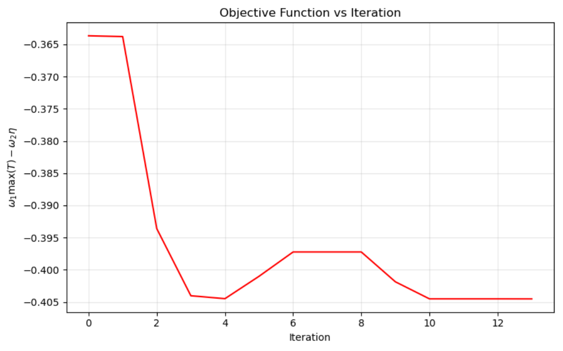
Table 1: Summary of NumPy FD optimization results (single weighting case).

	v [m/s]	a [W/m^4]	b [W/m^4]	c [W/m^3]	$T_{\max}$ [°C]	η
Initial guess	15.0	10.0	10.0	156250	–	–
Optimum (FD)	20.0	-48.7	-48.7	$1.53 \cdot 10^5$	48.45	0.80

The iteration history can be visualized using several plots.

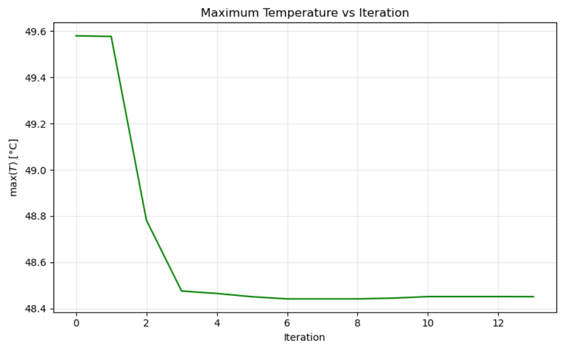


(a) Convergence of Lagrangian

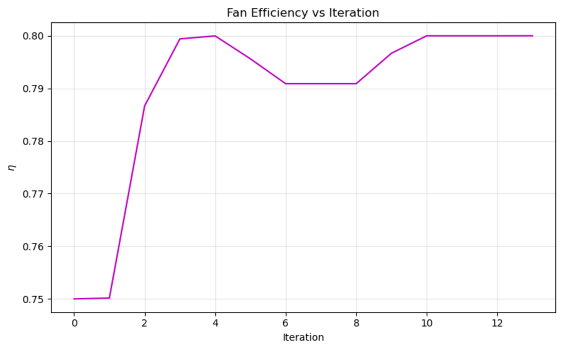


(b) Convergence of objective J

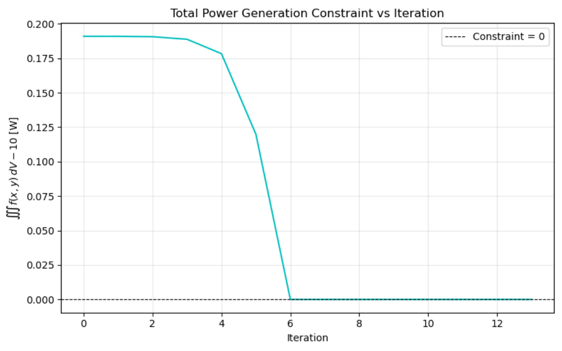
Figure 3: Convergence of the norm of the Lagrangian and the objective function for the NumPy/FD `trust-constr`. Both decrease over the course of the optimization, although the decrease is not perfectly smooth because the trust region and barrier parameters are being adjusted.



(a) Convergence of maximum temperature $T_{\max}$

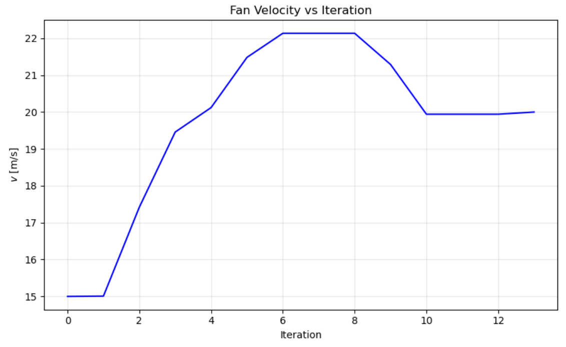


(b) Convergence of fan efficiency η

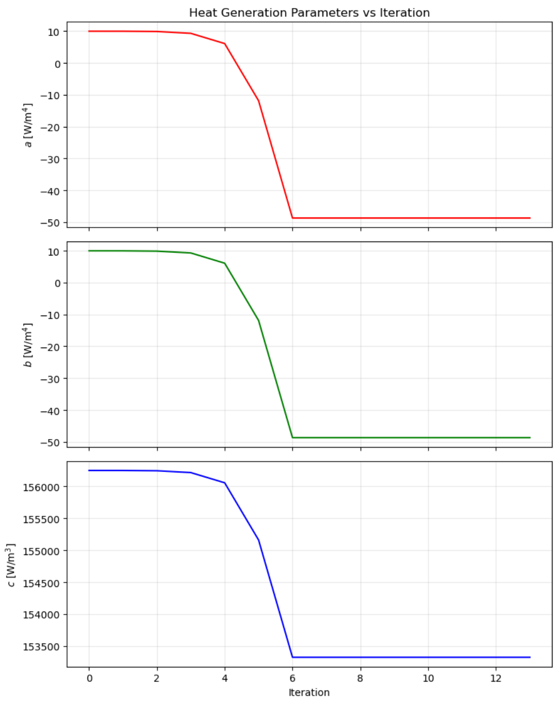


(c) Convergence of power generation constraint $g(a, b, c) = 0$

Figure 4: Convergence of individual objectives: max temperature, fan efficiency, and total power generation constraint. Temperature distribution decreases as fan efficiency reaches its maximum value. The constraint is satisfied early on in the optimization.



(a) Convergence of fan velocity v



(b) Convergence of

Figure 5: Convergence of the individual design variables (v, a, b, c) in the NumPy/FD optimization. The power-density parameters a, b, and c move quickly to enforce the total power constraint, while the fan velocity v converges more slowly towards its optimal value at 20 m/s.

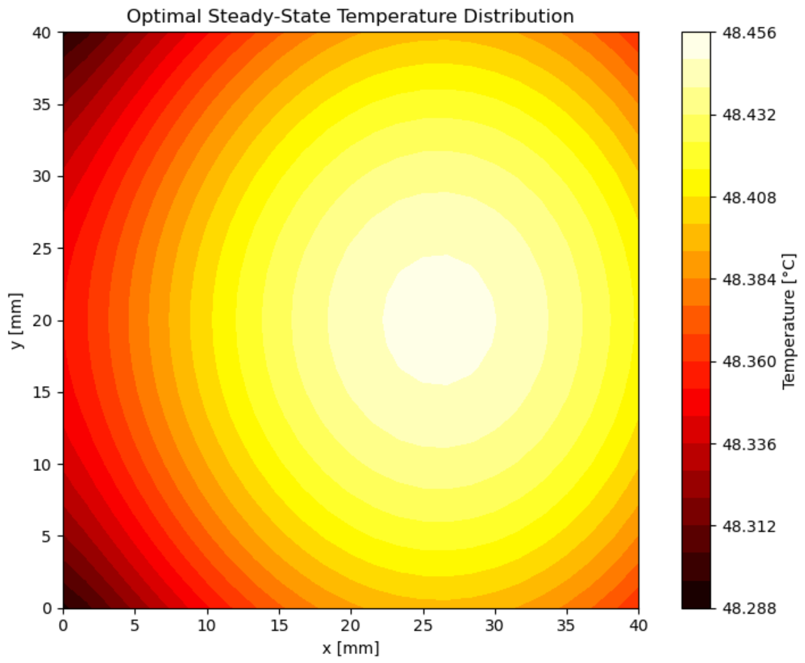


Figure 6: Optimal steady-state temperature distribution on the CPU for the NumPy/FD solution. The maximum temperature is slightly off-center, but because conduction within the CPU is strong, the spread between the maximum and minimum temperatures on the chip is small.

4 JAX Implementation and Automatic Differentiation

4.1 Making the PDE solver differentiable

To enable automatic differentiation, the original class-based solver was reimplemented in a purely functional style using `jax.numpy`. The key steps were:

- All parameters, discretizations, and constants are explicitly defined in the code
- All data (temperature field, total power generation, boundary conditions) are passed explicitly as arrays rather than stored in mutable object attributes.
- The explicit time-stepping loop is implemented using `jax.lax.scan`, which JAX can differentiate through efficiently.
- The steady-state temperature T_{ss} is obtained by iterating the time step a fixed number of times or until a convergence criterion is met inside a JAX-compatible loop.

The objective $J(\mathbf{x})$ and constraint $g(a,b,c)$ are wrapped as JAX functions. Automatic differentiation is then used to compute the gradient $\nabla J(\mathbf{x})$ for the objective and the constraint Jacobian $\nabla g(\mathbf{x})$.

It is important to note that the provided heat equation code for the optimization is quite small relative to codes defining more complicated system. It is the exception when a code can be reworked in a AD differentiable format within a short period of time. For codes that are developed over years this is impractical and likely impossible in most cases. In these situations, better derivates can be found with methods like complex step to find optimal divisors for FD or high fidelity surrogate models can be fit to the outputs of the code and then easily differentiated through to get and AD. In this particular case, the code is reworked.

4.2 Verifying the steady-state JAX solution

Since the JAX implementation changes how the time stepping is written, it is important to verify that the JAX solver converges to the same steady-state temperature field as the original NumPy code. To do this, several time steps were chosen, and the difference between the final temperature fields was computed. This was done for a range of design variables between the initial condition and the optimum found with the Numpy/FD implementation.

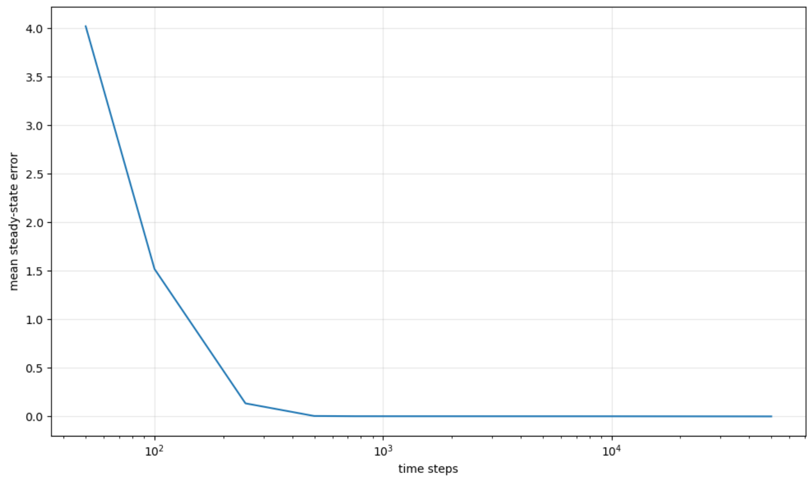


Figure 7: mean error between final temperature fields of conditions between initial and optimum as a function of the total number of time steps on a log scale. The error decreases and then flattens at zero indicating steady state is reached on average for a sufficient number of iterations

The convergence plot shows that the mean error becomes negligible after roughly 500 time steps (which was used as the time step value in implementation). For robustness of the gradient computation, the CFL number was reduced from 0.5 to 0.25 in the JAX implementation, slightly increasing the number of time steps but improving numerical stability for AD.

4.3 Differentiating the Objective Function

Conceptually, the sensitivity of the objective to the design variables is given by the total derivative

$$\frac{dJ}{dx} = \frac{\partial J}{\partial x} + \frac{\partial J}{\partial y}\frac{dy}{dx}, \tag{19}$$

where y represents all degrees of freedom of the steady-state temperature field. The first term captures any explicit dependence of the objective on the design variables (e.g., the explicit v-dependence of $\eta(v)$), and the second term captures the implicit dependence through the PDE constraint.

The derivative dy/dx can be expressed via the of the discretized PDE, $R(y, x) = 0$, as

$$\frac{dy}{dx} = -\left(\frac{\partial R}{\partial y}\right)^{-1}\frac{\partial R}{\partial x}, \tag{20}$$

which is the standard direct sensitivity method. JAX, through automatic differentiation, carries out a large chain rule through the code that effectively solves for dy/dx, avoiding the need to explicitly form or invert $\partial R/\partial y$.

For comparison, central finite differences approximate partial derivatives using

$$\frac{\partial J}{\partial x_i} \approx \frac{J(x + he_i) - J(x - he_i)}{2h}, \tag{21}$$

where e_i is the ith basis vector and h is a small perturbation. Very small h values lead to subtractive cancellation; larger h values introduce truncation error. There is typically an optimal h that balances these two effects.

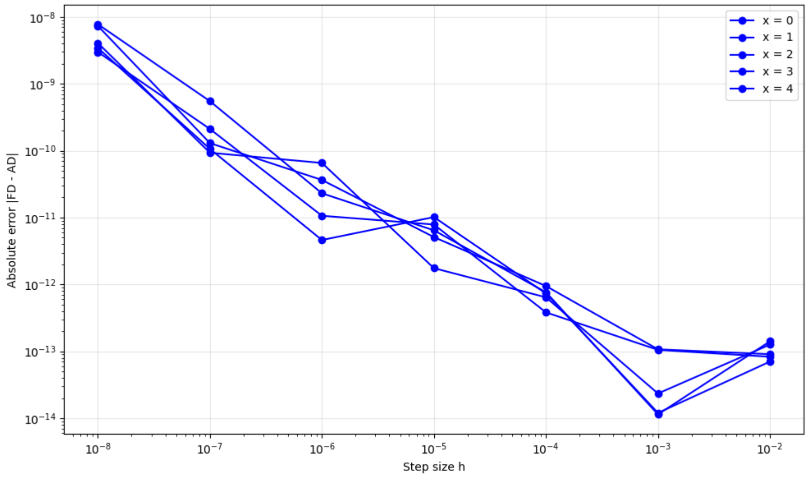


Figure 8: Log–log plot of the error between the JAX AD derivative $\partial J/\partial v$ and finite-difference (FD) approximations for a range of step sizes h. An intermediate step size $h \approx 10^{-3}$ minimizes the FD error; smaller h values are dominated by subtractive cancellation.

The comparison shows that a finite-difference step size of approximately $h = 10^{-3}$ minimizes the discrepancy between AD and FD for the most sensitive variable v. SciPy's default FD step size is slightly smaller, but in this problem the FD derivatives are still quite accurate across the optimization path. This explains why the original NumPy/FD optimization converges to essentially the same solution as the JAX/AD-based method.

4.4 Sensitivity Analysis

The AD and FD Jacobians were used to characterize how the objective and constraint respond to each design variable along the optimization path (linspace between x_0 and x^*). Table 2 summarizes the *mean* AD sensitivities of the objective function with respect to each design variable.

Table 2: Mean AD sensitivities of the objective J to each design variable over the NumPy optimization path.

Parameter	Mean $\partial J/\partial(\cdot)$
v (velocity)	-1.00×10^{-2}
a (heat coeff.)	3.83×10^{-11}
b (heat coeff.)	3.40×10^{-11}
c (heat coeff.)	1.70×10^{-9}

Similarly, Table 3 shows the mean AD sensitivities of the power constraint $g(x) = P(x) - 10$.

Table 3: Mean AD sensitivities of the constraint $g(x) = P(x) - 10$ to each design variable.

Parameter	Mean $\partial g/\partial(\cdot)$
v (velocity)	0.0
a (heat coeff.)	1.30×10^{-6}
b (heat coeff.)	1.30×10^{-6}
c (heat coeff.)	6.52×10^{-5}

These results show that the objective is dominated by the sensitivity to the fan velocity v. Changes in a, b, and c move the objective with magnitudes several orders of magnitude smaller than v (at least along the optimum-seeking trajectory). The constraint $P(x) - 10$ is dominated by the sensitivity to c, with much smaller dependence on a and b and essentially no dependence on v.

From the derivative of the efficiency,

$$\frac{d\eta}{dv} = -\frac{4}{100}v + \frac{8}{100} = 0.08 - 0.04\,v,$$

we can already anticipate that the optimizer will push v to the vicinity of 20 m/s, where η is maximized, and that other directions have comparatively weak influence on J.

This sensitivity structure implies that the problem is effectively low-dimensional: one could almost ignore a and b and reduce the design space to (v, c) without significantly changing the optimum. In a more realistic design problem, one might instead fix the total power output and optimize the distribution of circuit components to minimize manufacturing cost, or reliability risk, so that (a, b, c) would play a more prominent role in the objective.

5 Optimization with AD and variable scaling

The JAX implementation provides exact (up to numerical precision) Jacobians for both the objective and the constraint. These derivatives are supplied to `scipy.optimize.minimize` as an additional element in the constraint dictionary and as a direct argument in the function.

5.1 Scaling of design variables

In an initial attempt, using exact JAX gradients together with the default quasi-Newton Hessian approximation in `trust-constr` led to slow convergence and warnings about the Hessian update (e.g., `delta_grad == 0`). This behaviour was linked to poor scaling of the design variables and near-linear structure in the problem.

To improve conditioning, the design variables were non-dimensionalized:

$$\tilde{v} = \frac{v}{20}, \qquad \tilde{a} = \frac{a}{50}, \qquad \tilde{b} = \frac{b}{50}, \qquad \tilde{c} = \frac{c}{10^5}. \tag{22}$$

The optimizer works in the scaled variables $\tilde{\mathbf{x}} = [\tilde{v}, \tilde{a}, \tilde{b}, \tilde{c}]^\top$, and the physical variables are reconstructed inside the objective and constraint functions. This brings all design coordinates to $\mathcal{O}(1)$, which stabilizes the trust-region Hessian update.

5.2 AD-based scaled optimization results

With scaling and JAX-based derivatives, `trust-constr` converges reliably. Using the same weights $\omega_1 = 0.2$, $\omega_2 = 0.8$ and an initial guess (essentially the same as the numpy/FD IC in the unitless framework)

$$v_0 = 0.5 \text{ m/s}, \quad a_0 = 0.1, \quad b_0 = 0.1, \quad c_0 = 1.0,$$

the optimizer quickly increases v to the neighbourhood of 20 m/s and adjusts c to satisfy the power constraint. The design parameters a and b do not move from their initial values and effectively remain at $a \approx 0.1$, $b \approx 0.1$ in unitless form.

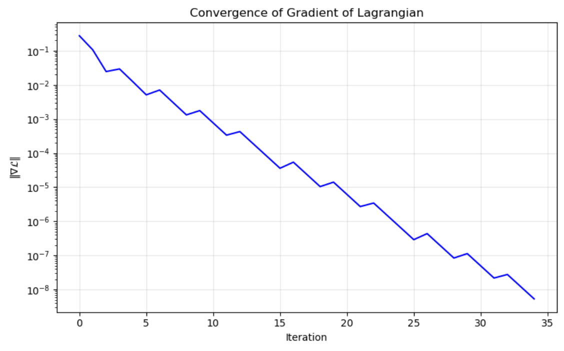
The final solution in terms of the physical units is

$$\begin{aligned} v^* &\approx 20.0 \text{ m/s}, \\ a^* &\approx 5.0 \text{ W/m}^4, \\ b^* &\approx 5.0 \text{ W/m}^4, \\ c^* &\approx 1.53 \times 10^5 \text{ W/m}^3, \end{aligned}$$

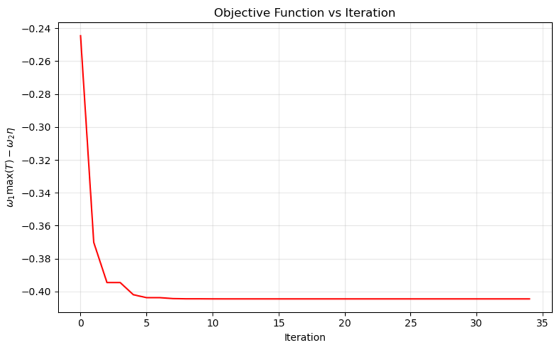
with

$$T_{\max} \approx 321.45 \text{ K } (48.45°\text{C}), \qquad \eta(v^*) \approx 0.80.$$

These objective components agree with the NumPy FD solution, even though the optimal (a, b) differ. The iteration history can be visualized using several plots:

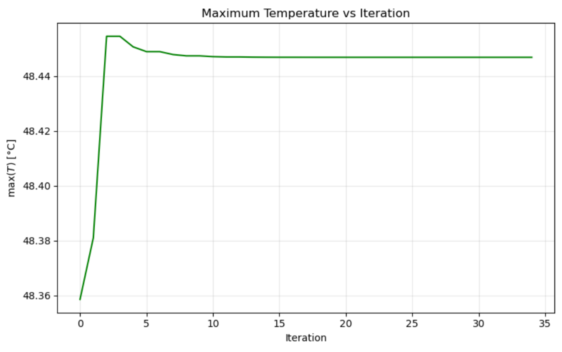


(a) Convergence of Lagrangian

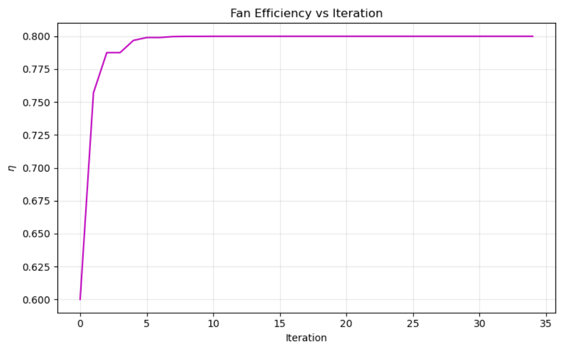


(b) Convergence of objective J

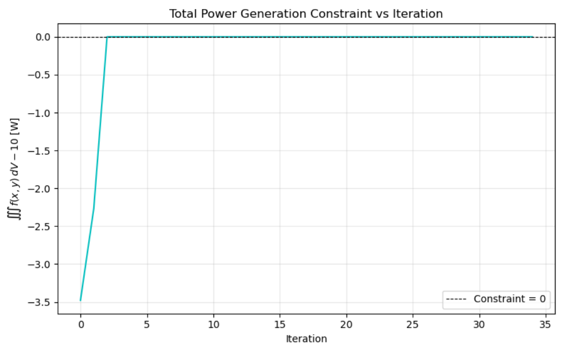
Figure 9: Scaled JAX/AD `trust-constr` convergence. The Lagrangian decreases roughly linearly on a log-scale, while the objective drops quickly in the first few iterations and then plateaus.



(a) Convergence of maximum temperature $T_{\max}$

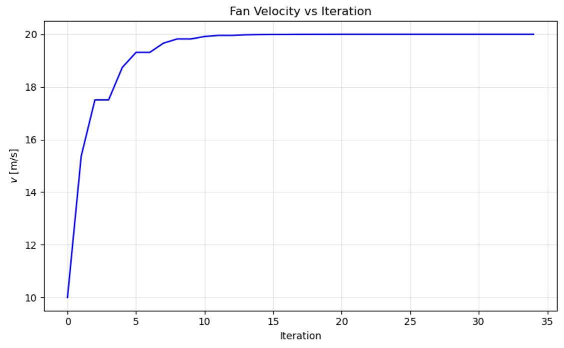


(b) Convergence of fan efficiency η

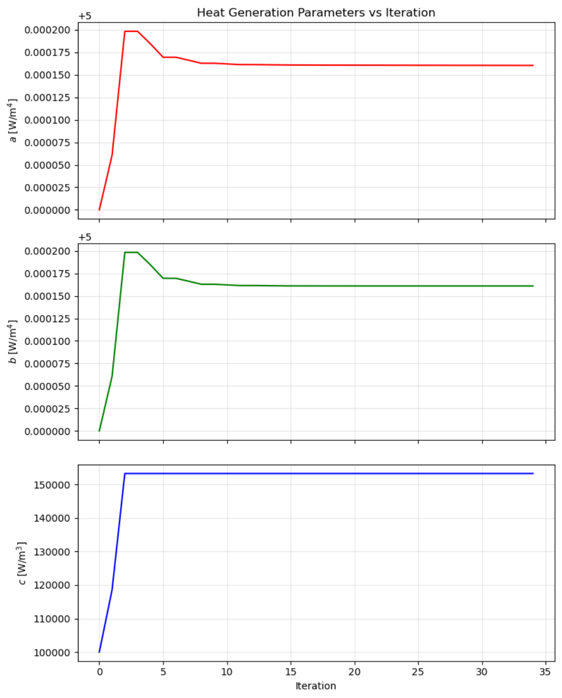


(c) Convergence of power generation constraint $g(a,b,c) = 0$

Figure 10: Convergence of individual objectives: max temperature, fan efficiency, and total power generation constraint. The power constraint is satisfied within the first few iterations, and η converges to its maximum value then plateaus. Similar behaviour for the max temperature



(a) Convergence of fan velocity v



(b) Convergence of

Figure 11: Convergence of the design variables in the scaled JAX/AD optimization. The coefficient c is rapidly driven to a value that satisfies the power constraint. The coefficients a and b change very little, while v converges more slowly and then plateaus near 20 m/s.

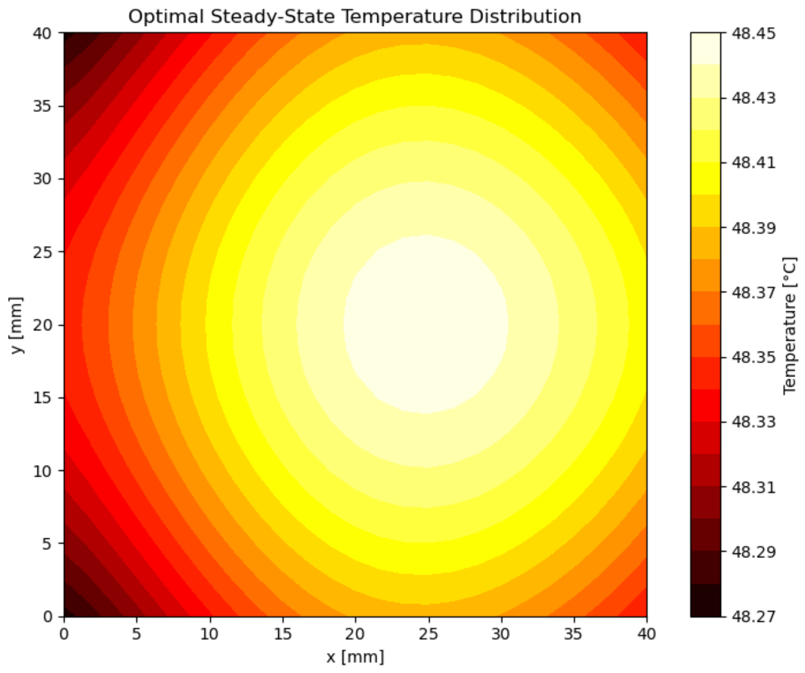


Figure 12: Optimal steady-state temperature distribution from the scaled JAX/AD optimization. The shape is essentially the same as in the NumPy/FD solution; small differences near the contour edges.

5.3 Flat directions and KKT perspective

From a KKT/sensitivity perspective, the behavior of the JAX-based optimization reveals that the coefficients a and b span an almost "flat" subspace of the design space. Denote the Lagrangian by

$$\mathcal{L}(x, \lambda) = J(x) + \lambda\, g(x),$$

where $g(x) = P(x) - 10$ is the equality constraint. At an interior constrained optimum $(x^\star, \lambda^\star)$, the first-order KKT conditions require

$$\nabla_x \mathcal{L}(x^\star, \lambda^\star) = 0, \qquad g(x^\star) = 0. \tag{23}$$

The AD sensitivities show that, near the optimum, the components of $\nabla_x \mathcal{L}$ in the a- and b-directions are several orders of magnitude smaller than the components in the v- and c-directions. Physically, adjusting the linear shape of the heat source (via a and b) only weakly perturbs both the total heat generation and the maximum temperature, compared to changing the uniform term c or the fan velocity v.

In the KKT system, this means that along the feasible manifold $g(x) = 0$ there is an almost flat valley in the (a, b) plane: small moves in (a, b) leave both $J(x)$ and $g(x)$ essentially unchanged. Because heat conduction in the CPU is relatively strong and the total heat generation is fixed, redistributing heat generation via (a, b) has only a weak effect on the final maximum temperature. Numerically, the AD gradients in the a and b directions are extremely small once c has been adjusted to satisfy the power constraint:

$$\left|\frac{\partial J}{\partial a}\right| \approx 0, \qquad \left|\frac{\partial J}{\partial b}\right| \approx 0. \tag{24}$$

Consequently, the KKT stationarity conditions in the a and b components are dominated by near-zero contributions from both $\partial J/\partial a$ and $\lambda\, \partial g/\partial a$.

Any point on this nearly flat manifold satisfies the KKT conditions to within numerical tolerance, so the optimizer has no strong incentive to move a and b away from their initial values. The larger changes in a and b observed in the original NumPy/FD run are therefore best interpreted as motion driven by finite-difference noise and poor scaling in directions of very weak sensitivity, rather than evidence of a significantly better design.

5.4 Comparing AD and FD Scaled Optimization

Scaling the inputs to the objective and constraint for the Numpy/FD implementation leads to the static a, b behaviour as observed with JAX/AD. The results for the two scaled optimizations are presented in table 4

Table 4: Comparison of optimization statistics for AD (JAX) and FD (NumPy) runs.

	AD (JAX, scaled)	FD (NumPy)
Iterations	35	12
Function evaluations	26	35
Jacobian evaluations	121	7 (FD)
Final design $[v, a, b, c]$	$[1.0,\ 0.1,\ 0.1,\ 1.5332]$ (scaled)	$[1.0,\ 0.1,\ 0.1,\ 1.5332]$ (scaled)
	$\Rightarrow [20,\ 5,\ 5,\ 1.5332 \times 10^5]$	$\Rightarrow [20,\ 5,\ 5,\ 1.5332 \times 10^5]$

The JAX implementation provides two main benefits:

- Exact gradients reduce the number of expensive PDE solves per iteration compared with FD-based Jacobians. This decreases the total time of the optimization.
- The derivative information reveals gives credence to what range the FD derivative is trust worthy. This Jacobian confidence important structure in the problem, particularly the presence of nearly flat directions in (a, b).

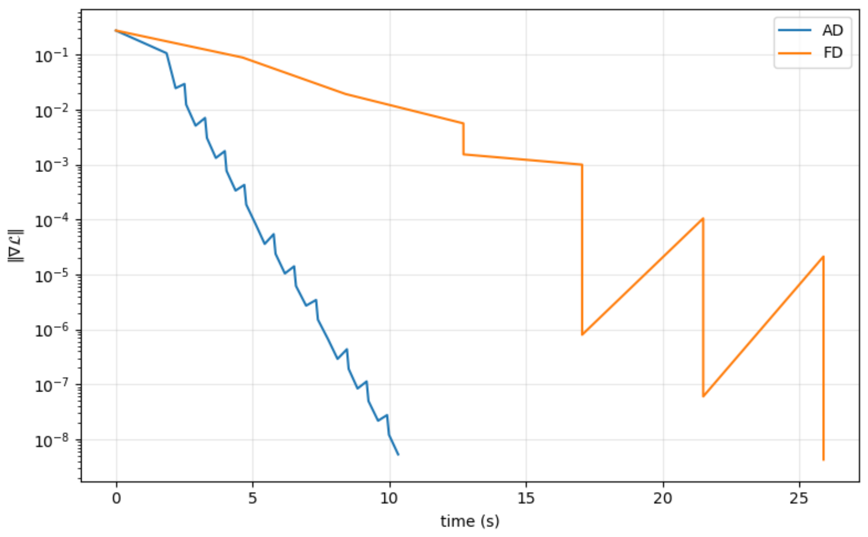


Figure 13: Convergence of the Lagrangian with respect to computation time (s) for JAX/AD and Numpy/FD. The differentiation through the the PDE is faster that multiple recomputation with perturbations of the design variables

6 Multi-Objective Perspective and Pareto Analysis

To explore the trade-off between maximum temperature and fan efficiency, the optimization was repeated for different weight pairs (ω_1, ω_2). Three representative cases are:

- case A: $(\omega_1, \omega_2) = (0.001, 0.999)$ (strong emphasis on efficiency);
- case B: $(\omega_1, \omega_2) = (0.5, 0.5)$ (balanced weights);
- case C: $(\omega_1, \omega_2) = (0.999, 0.001)$ (strong emphasis on temperature).

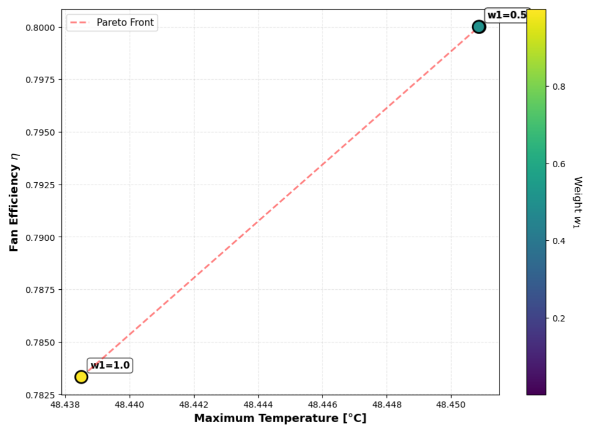
A summary of the corresponding optimal designs (using the AD implementation) is:

Table 5: Representative points on the Pareto front (AD-based optimization).

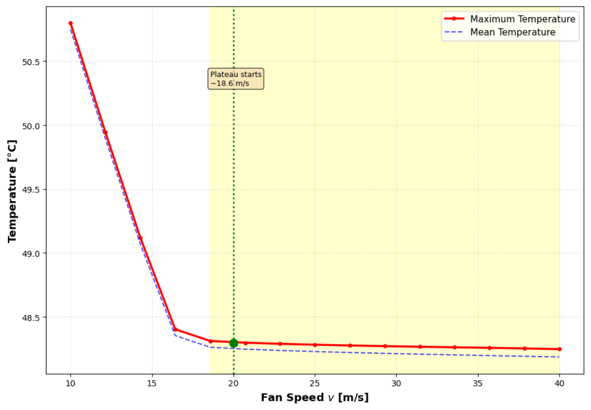
ω_1	ω_2	$T_{\max}$ [°C]	η	v [m/s]
0.001	0.999	48.45	0.8000	20.00
0.5	0.5	48.45	0.8000	20.00
0.999	0.001	48.44	0.7833	22.89

The Pareto results show that, despite drastically different weights, the optimizer tends to select fan velocities near the efficiency optimum $v \approx 20$ m/s. The corresponding maximum temperature plateaus around 48.45°C; further increasing v from this region yields only marginal additional cooling but requires much more fan power. This plateau is a direct consequence of the diminishing returns of convective heat transfer (through the Reynolds and Prandtl dependence of the convective coefficient) combined with strong conduction within the CPU.

The plot of $T_{\max}$ as a function of v exposes this behavior clearly: temperature decreases rapidly at low velocities, but then flattens out around the optimum, explaining why the sensitivity of the objective is dominated by v in this region. The optimizer locks onto the fan efficiency peak finding that no further temperature gains can be made and uses c primarily to enforce the power constraint, with a and b playing a minimal role.



(a) Pareto front in $(T_{\max}, \eta)$ space



(b) $T_{\max}$ versus v

Figure 14: Multi-objective behavior of the CPU cooling design. Left: Pareto front formed by varying (w_1, w_2). Right: maximum steady-state temperature as a function of fan velocity.

7 Conclusions

This project applied numerical optimization and automatic differentiation to a multi-objective, PDE-constrained design problem: cooling a CPU using a fan. The main steps and findings are:

- A transient finite-difference solver for the 2D heat equation was used to compute steady-state temperature distributions for given design variables. This solver was first implemented in NumPy and then recast in JAX to allow differentiation through the PDE solve.

- The design problem was formulated as a multi-objective optimization with an equality constraint on total power and bound constraints on the design variables. The two objectives were the maximum steady-state temperature and a quadratic fan-efficiency metric.

- The baseline NumPy implementation, using FD approximations for the Jacobian, converged reliably because the FD approximations were generally quite accurate for this problem. However it required a relatively large long computation time because each gradient evaluation demanded multiple PDE solves.

- The JAX implementation provided exact gradients of both the objective and the constraint, substantially reducing the run time. Appropriate scaling of the design variables was crucial for robust performance of the `trust-constr` method.

- Sensitivity analysis and the KKT conditions revealed nearly flat directions in the design space, especially in the (a, b) parameters that control the spatial distribution of heat generation. These flat directions explain why the FD and AD implementations converge to different (a, b) values but essentially identical objective values.
- A simple Pareto analysis showed that, for most reasonable weightings of temperature and efficiency, the optimal design uses a fan speed close to $20\,\mathrm{m/s}$, where the efficiency peaks and the temperature approaches a plateau set by conduction.

From an engineering perspective, the study illustrates how:

1. automatic differentiation gives can be integrated into PDE-based design problems by restructuring the solver in a differentiable manner. Accurate derivatives are very powerful for analysis; however, implementation of AD will only work for smaller code libraries.
2. sensitivity analysis can reveal low-dimensional active subspaces and justify simplifying assumptions, such as treating some design variables as effectively fixed.

A more realistic CPU design problem could enrich this framework by including additional objectives (e.g., manufacturing cost associated with (a, b, c)), more complex heat-generation models, or non-symmetric geometries where (a, b) have a stronger influence on both the constraint and the maximum temperature.

Appendix: Steady-State Formulation and AD Efficiency

In this project, the steady-state temperature distribution is obtained by integrating the transient heat equation forward in time using an explicit scheme until a convergence criterion is met. From an automatic differentiation (AD) perspective, this means that each evaluation of the objective $J(x)$ builds a computation graph that includes *every* time step:

$$T^0 \rightarrow T^1 \rightarrow T^2 \rightarrow \cdots \rightarrow T^N = T_\infty.$$

Reverse-mode AD must then backpropagate through this entire chain. The cost of computing $\nabla J(x)$ therefore scales with the number of time steps N.

If instead the steady-state temperature were computed by directly solving the steady-state PDE,

$$-k\left(\frac{\partial^2 T_{ss}}{\partial x^2} + \frac{\partial^2 T_{ss}}{\partial y^2}\right) = \dot{E}(x, y; a, b, c), \tag{25}$$

then, after spatial discretization, the problem would reduce to a linear algebra system

$$A(\theta)\, T_\infty(\theta) = f(\theta), \tag{26}$$

where $\theta = [v, a, b, c]^\top$ collects the design variables, $A(\theta)$ encodes diffusion and boundary conditions, and $f(\theta)$ encodes the heat generation. In this formulation, the map

$$\theta \longmapsto T_{ss}(\theta)$$

is implemented by a single linear solve (e.g. a direct solver or an iterative method run to convergence), rather than hundreds of explicit time steps. This makes each gradient evaluation significantly more efficient, especially when the transient solver must take many small time steps to reach steady state.