

Supplementary Information:

Sufficient is better than optimal for training neural networks

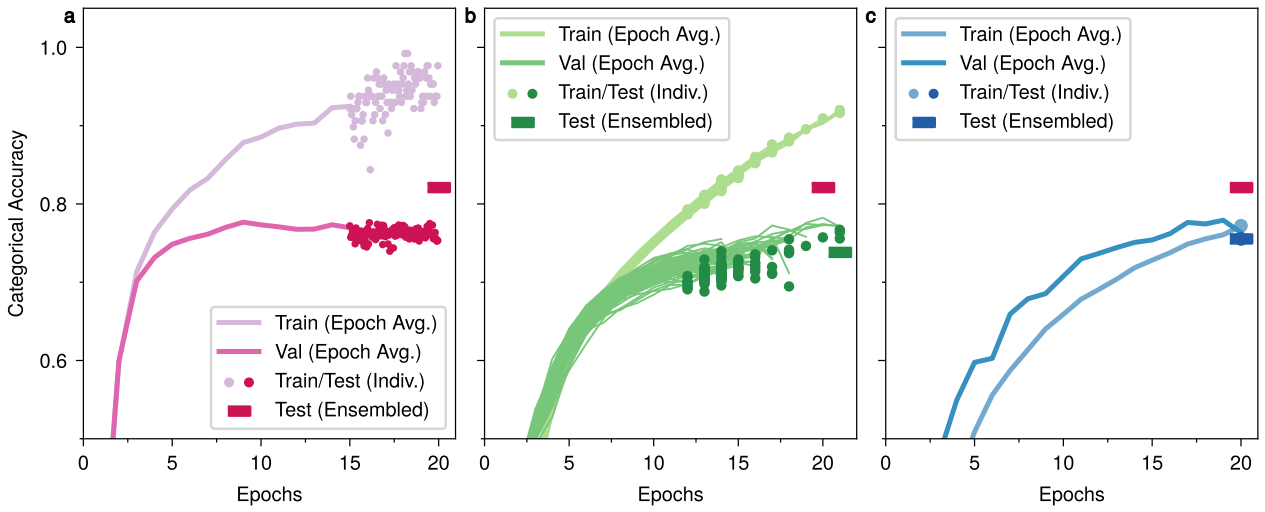
Irina Babayan,^{1†} Hazhir Aliahmadi^{1†}, and Greg van Anders^{1*}

¹ Department of Physics, Engineering Physics, and Astronomy, Queen's University, Kingston ON, K7L 3N6, Canada

[†]These authors contributed equally. * Email: gva@queensu.ca

Supplementary Figures

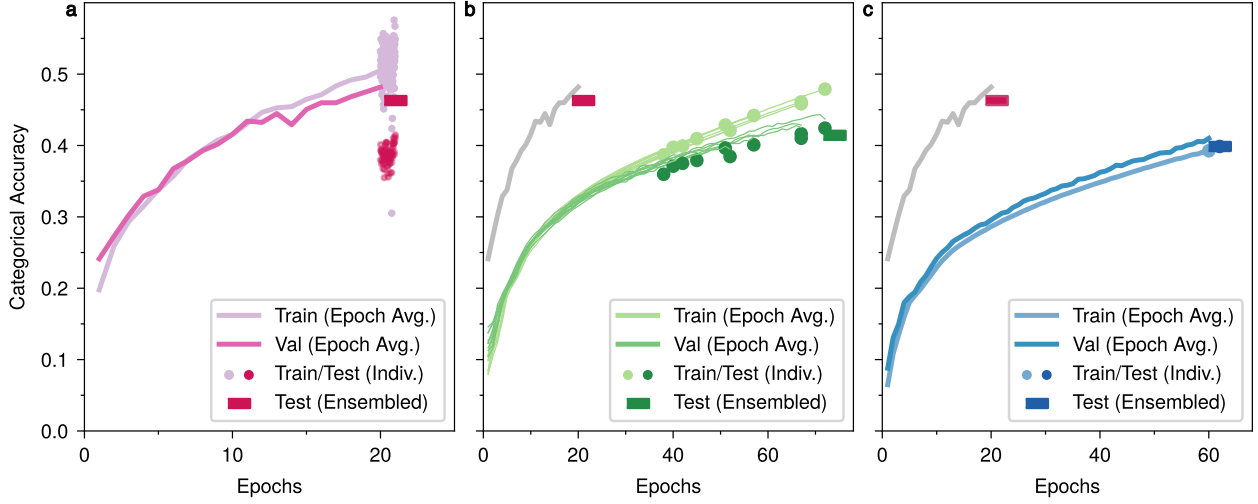
CIFAR-10 and Translation Training Convergence



Supplementary Figure 1. The training convergence curves corresponding to the CIFAR-10 training comparison results, also shown in Main Fig. 3, show that simmering (panel a) produces both the highest ensemble accuracy (red vs. green or blue rectangular marker, panels b and c respectively) and the most significant ensembling advantage (red round vs. rectangular markers in panel a) compared to early stopping (green round vs. rectangular markers in panel b), and dropout (blue round vs. rectangular markers in panel c). In all panels, lines denote the individual model accuracy averaged over all weight updates of a given epoch, and round markers denote the performance of individual models that are included in the ultimate ensemble.

Supplementary Figure 1 shows the training performance curves that accompany the CIFAR-10 test accuracy results presented in Main Fig. 3. Simmering (Supplementary Fig. 1a) and early stopping (Supplementary Fig. 1b) converge to similar individual-model training and validation data accuracies, and their ensemble predictions are aggregated in the same manner (majority voting). However, the early stopped ensemble aggregate prediction accuracy is approximately the average accuracy of its constituent models (round vs. rectangular markers in Supplementary Fig. 1b), whereas the simmering ensemble significantly improves on its constituent models' accuracies (round vs. rectangular markers in Supplementary Fig. 1a). The model trained with dropout (Supplementary Fig. 1c) outperforms the early stopping ensemble but does not exceed the ensembled test accuracy of simmering for equivalent training time. Though all three model ensembles have similar individual-model epoch-averaged validation and test accuracies, the ensemble collected via simmering yields the highest ensembled prediction accuracy.

Supplementary Figure 2 shows the accuracy convergence curves that accompany the PT-EN translation test accuracy

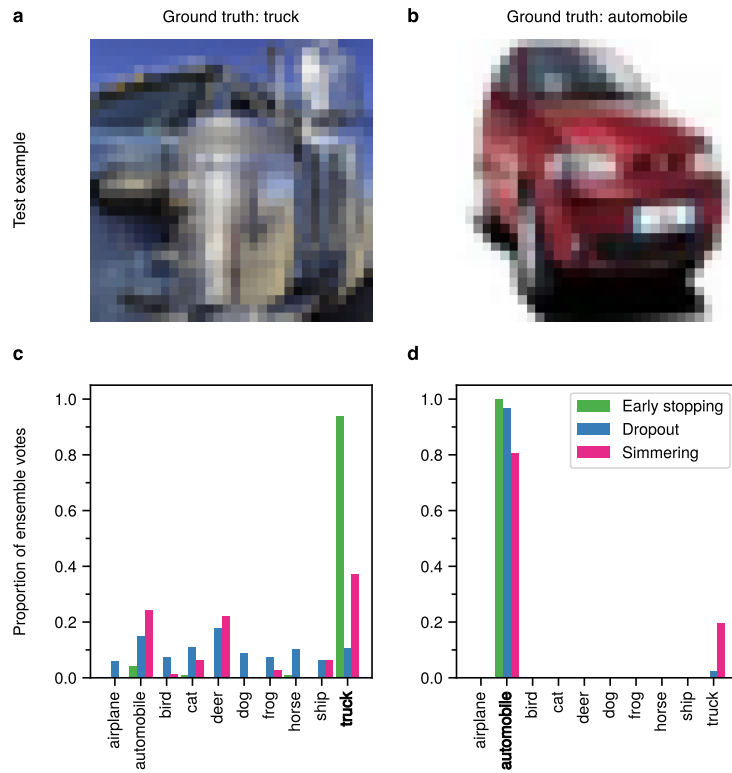


Supplementary Figure 2. The training convergence curves corresponding to the PT-EN translation training comparison results, also shown in Main Fig. 3, show that simmering (panel a) produces both the highest ensemble accuracy (red rectangular marker, all panels) and the most significant ensembling advantage (round vs. rectangular markers in panel a) in less than half of the training time (grey vs. coloured line, and red vs. green or blue rectangular marker in panels b and c) compared to training with an equivalent learning rate with early stopping (green lines and markers, panel b), and dropout (blue lines and markers, panel c). In all panels, lines denote individual-model accuracy averaged over all weight updates of a given epoch, and round markers denote the performance of individual models that are included in the ultimate ensemble. The simmering epoch-averaged validation curve and ensembled test accuracy (grey line and red rectangular marker, respectively, in panels b and c) are provided for convergence timescale and performance comparison.

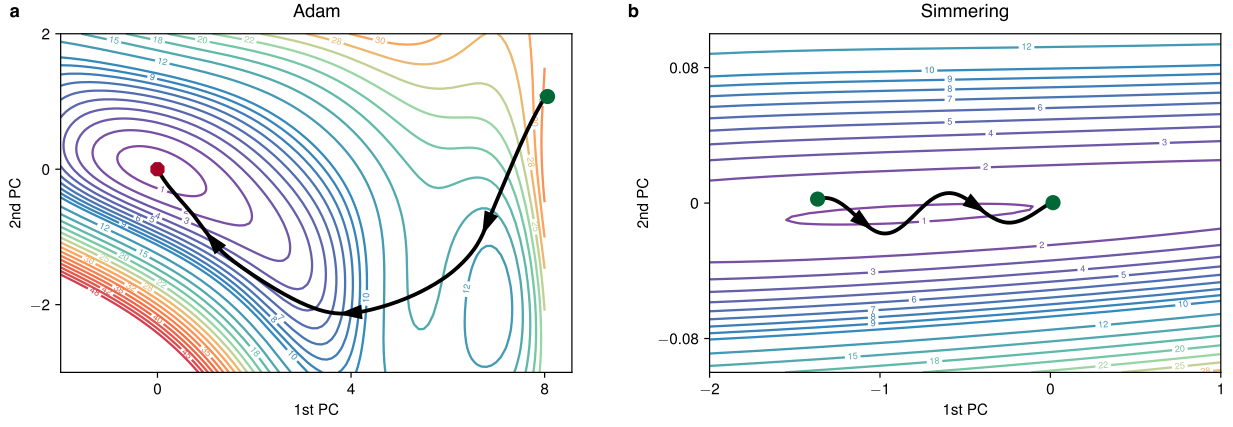
results (rectangular markers in Supplementary Fig. 2) shown in Main Fig. 3. Given an identical, fixed learning rate, simmering converges to a higher individual-model training and validation accuracy than models trained with early stopping and dropout in less than half of the number of epochs. As for CIFAR-10, simmering also displays the greatest ensembling advantage relative to the test accuracy of ensemble members: the test accuracy of the early stopped ensemble is lower than its most accurate constituent model (green round marker vs. green rectangular marker in Supplementary Fig. 2), whereas the simmering ensemble accuracy is several percentage points higher than the most accurate constituent model. Dropout converges the most slowly out of the three methods tested (red vs. blue rectangular markers in Supplementary Fig. 2). Overall, given identical learning rates, simmering converges to a higher ensembled test accuracy than both an ensemble of early stopped models and a model trained with dropout in a fraction of the number of training epochs.

CIFAR-10 Prediction Distribution Comparison

Beyond generalizable prediction, simmering also yields prediction uncertainty estimates that can reflect data difficulty. Supplementary Fig. 3 shows how the three ensemble learning approaches predict two different CIFAR-10 test samples (Supplementary Fig. 3a–b) chosen based on relative image difficulty (pixel-intensity standard deviation and Shannon entropy). Though ensemble uncertainty estimates can be extracted either directly or indirectly (see Supplementary Methods) from all three ensemble learning approaches, the ensemble generated by ab initio simmering allows for uncertainty estimates that reflect relative image difficulty as shown in Supplementary Fig. 3c–d. Thus, the maximum entropy distribution sampling that supports simmering provides generalizable predictions and may offer additional insight into the neural network’s learning process.



Supplementary Figure 3. Ab initio simmering correctly classifies CIFAR-10 test images. Panel b shows an image with high pixel-intensity standard deviation and low Shannon entropy, which depicts an automobile, with a clear foreground and with the automobile in full view. Early stopping (green bars), dropout (blue bars) and ab initio simmering (pink bars) ensembles yield predictions with similarly high confidence (high proportion of ensemble votes). In contrast, panel a shows an image with low pixel-intensity standard deviation and high Shannon entropy, which depicts a truck at close range with low background contrast. Early stopping remains nearly equally confident (panel c, green bar) whereas simmering predicts the correct class with lower confidence, aligning with the difficulty of identifying the truck in the image. Simmering’s second-most voted prediction bears relation to the correct class (“automobile” and “truck” have similar image features). Though dropout also predicts uncertainly, it predicts the correct label with near-equivalent confidence to most other classes. Thus, ab initio simmering predicts images of varying difficulty accurately with confidence that can reflect image difficulty.



Supplementary Figure 4. Simmering (panel b), an implementation of sufficient training, produces a distinct training trajectory (black curve, panel b) from the Adam optimizer (black curve, panel a). The Adam optimizer (black curve with arrow indicating direction of traversal, panel a) traverses the training loss landscape (coloured contours in panel a) perpendicular to the loss contours, as expected in gradient-based optimization. In contrast, simmering travels along equivalent-loss (coloured contours in panel b) directions in parameter space, aligning with the information-geometric interpretation of sufficient training presented in Main Methods. The Adam trajectory (panel a) corresponds to the entire Adam training phase of Main Fig. 1a (beginning and end marked by the green and red markers respectively, panel a), and the simmering trajectory is a 100-epoch slice (endpoints indicated by green markers, panel b) of the simmering sampling (after the target temperature is reached) shown in Fig. 1a. The loss landscape and training trajectories have been projected onto the principal components (PCs) of the Adam and simmering training trajectories in panels a and b respectively.

Training Trajectory Visualization

Figure 4, generated using the loss visualization method of Ref.¹, shows the difference in training dynamics between Adam,² a gradient-based optimizer, and simmering, an implementation of sufficient training. The Adam optimizer travels perpendicular to the loss contours, whereas simmering travels parallel to the direction of near-equivalent loss. The first principal component (PC) of Adam describes 90.44% of its trajectory variance in parameter space, and the second PC describes another 6.12%. For simmering, the first and second PCs describe 99.9428% and 0.0324% of the parameter space trajectory respectively. In the latter case, a larger proportion of the training trajectory is described by the first PC. Since this PC points along the direction of equivalent loss in parameter space, Supplementary Fig. 4b shows that sufficient training automatically identifies and samples along sloppy (equivalent-loss) modes in model parameter space as described in Main Methods. Note that optimization-based training can also exhibit a form of oscillation that results from finite differencing artifacts, whereas simmering is designed to exhibit thermal oscillations instead. These thermal oscillations are a critical feature that simmering leverages to sample an ensemble of models.

Supplementary Methods

Simmering Notation

Supplementary Table 1 describes the notation used in Main Methods and Supplementary Methods. For each symbol, we describe its physical meaning (in statistical physics literature) and its equivalent or analogous notation in machine learning (ML) literature.

Symbol	Physical Terminology	Equivalent ML Terminology
$\mathbf{x} \in \mathbb{R}^N$	Full parameter vector (weights and biases)	θ (trainable parameters)
$L(\mathbf{x}, \mathcal{D})$	Training loss evaluated on data set $\mathcal{D}$	$\mathcal{L}(\theta; \mathcal{D})$
$T = 1/\beta$	Temperature	Temperature ^{3,4}
$p(\mathbf{x} \mathcal{D})$	Canonical distribution over parameters	Tempered posterior ⁴ over weights
$\theta(\mathbf{x}, \mathcal{D}) \in \mathbb{R}^n$	Collective variables / collective coordinates	Latent parameter; low-dimensional embedding of models ⁵
$e^{-\beta F(\theta, \mathcal{D})}$	Effective free energy	Energy in latent parameter space
$p(\theta \mathcal{D})$	Distribution over collective variables	Tempered posterior ⁴ over latent parameters
x_i, v_i, m_i	Real particle position, velocity, mass (particle index i)	Parameter value θ_t , momentum m_t^2 , per-param scale
s_k, v_{s_k}	Virtual thermostat particle position, velocity (chain index k)	Auxiliary thermostat states ξ , momentum $\mathbf{P}^6$
$a_i(t)$	Real-particle acceleration	Preconditioned / scaled gradient
$a_{s_1}(t)$	First thermostat acceleration	Temperature control term
$a_{s_k}(t)$	Chain thermostat accelerations ($k \geq 2$)	Coupled control terms
Q_k	Thermostat masses	Hyperparameters controlling response time
N_{NHC}	Length of the Nosé–Hoover chain	Thermostat depth
T_{target}	Target thermostat temperature	Set-point for temperature
T_{real}	Measured kinetic temperature	Empirical Temperature / Scaled Momentum Variance
Δt	Integration time step / learning rate	Step size η
$v_{i,0} \leftarrow \sqrt{T/m_i}$	Velocity initialization	Momentum initialization consistent with T^6
t, t_{max}	Iteration index and maximum iterations	Training step k / total steps K

Supplementary Table 1. Symbols used to describe simmering and its implementation in Methods and Supplementary Methods, and their analogues in machine learning (ML) contexts.

Numerical Implementation

To numerically integrate the particle positions over time, we discretize Main Equations 24–28 using a Verlet integrator for position and velocity for both the real and virtual particles. Verlet integration is used here because it preserves Hamilton’s equations.^{7,8} The resulting discretized equations of motion are

$$x_i(t + \Delta t/2) = x_i(t) + \frac{\Delta t}{2} v_i(t) \quad (1)$$

$$s_{2k}(t + \Delta t/2) = s_{2k}(t) + \frac{\Delta t}{2} v_{s_{2k}}(t) \quad (2)$$

$$v_{s_{2k-1}}(t + \Delta t/2) = v_{s_{2k-1}}(t) e^{\frac{-\Delta t}{2} v_{s_{2k}}} + \frac{\Delta t}{2} a_{s_{2k-1}}(t) e^{\frac{-\Delta t}{4} v_{s_{2k}}}, \quad (3)$$

$$v_i(t + \Delta t) = v_i(t) e^{-\Delta t v_{s_1}(t + \Delta t/2)} + \Delta t a_i(t + \Delta t/2) e^{-\frac{1}{2} \Delta t v_{s_1}(t + \Delta t/2)} \quad (4)$$

$$s_{2k-1}(t + \Delta t) = s_{2k-1}(t) + \Delta t v_{s_{2k-1}}(t + \Delta t/2) \quad (5)$$

$$v_{s_{2k}}(t + \Delta t) = v_{s_{2k}}(t) e^{-\Delta t v_{s_{2k+1}}(t + \Delta t/2)} + \Delta t a_{s_{2k}}(t + \Delta t/2) e^{-\frac{1}{2} \Delta t v_{s_{2k+1}}(t + \Delta t/2)}, \quad (6)$$

$$x_i(t + \Delta t) = x_i(t + \Delta t/2) + \frac{\Delta t}{2} v_i(t + \Delta t) \quad (7)$$

$$s_{2k}(t + \Delta t) = s_{2k}(t + \Delta t/2) + \frac{\Delta t}{2} v_{s_{2k}}(t + \Delta t) \quad (8)$$

$$v_{s_{2k-1}}(t + \Delta t) = v_{s_{2k-1}}(t + \Delta t/2) e^{-\frac{1}{2} \Delta t v_{s_{2k}}(t + \Delta t)} + \frac{\Delta t}{2} a_{s_{2k-1}}(t + \Delta t) e^{-\frac{1}{4} \Delta t v_{s_{2k}}(t + \Delta t)}, \quad (9)$$

where $a_i, a_1, a_{s_k, k \neq 1}$ are defined in Main Equations 29–31. Given this numerical integration scheme, an ensemble of neural networks can be generated by selecting an appropriate learning rate Δt and iteratively computing $x(t + \Delta t)$.

Training Procedure

Algorithm 1 describes the implementation of simmering used to generate the results presented in this work. There are two main stages of simmering: the equilibration stage, and the model sampling stage. The purpose of this algorithm is to sample models from the canonical distribution of models at a given temperature, and the equilibration stage ensures that the model parameter distribution reaches thermal equilibrium, i.e., is distributed according to the canonical distribution, before models are sampled. The main simmering hyperparameters are the thermostat temperature T , the particle mass m , and the learning rate Δt . The thermostat temperature T can be either a constant or a schedule that depends on training time t . Model sampling can commence once the average parameter temperature T_{real} over time is approximately equal to the thermostat temperature T . Note that the precision with which the neural network parameter temperature will fluctuate about T depends on the neural network parameter space dimensionality, so expect that T_{real} will fluctuate with a larger amplitude for neural networks with less parameters. For our training experiments, all real and virtual particles had the same mass, i.e., $m_i = Q_k \forall i, k$, based on Ref.⁹. Generally, decreasing the mass makes the parameter updates more sensitive to the loss gradient (see m dependence in line 20) and thus can accelerate equilibration. The learning rate Δt is fixed and identical for all particles (neural network parameters and NHC particles). We also note that we used a NHC length of 10 for all experiments.

To commence equilibration, provide the initial weights and biases of the neural network as $x_{all,0}$, and proceed with Algorithm 1. During training, before executing line 20 of the for-loop in Algorithm 1, set the neural network parameters

to $x_{all,t+1/2}$, and use backpropagation to compute the gradient of the loss $\nabla_x L$. Once equilibration is complete, reuse the same algorithm for sampling models. The only difference in execution is that the model parameters and/or predictions generated at the loss gradient computation stage (line 18 of Algorithm 1) are saved for offline ensemble analysis. There is no prescription for sampling frequency. However, some options for sampling frequency include saving every iteration’s predictions, saving the predictions at a fixed iteration interval, or saving randomly. If sampling at some fixed interval, check that the samples are uncorrelated at that interval. After the desired number of samples have been collected, the set of models and/or predictions can be aggregated or analyzed as a distribution of solutions in the ways described in the main text.

Computational Complexity, Training Time, and Resource Usage

Simmering has a per-iteration complexity comparable to standard first-order methods (e.g., SGD with momentum), with time complexity $O(N)$. The parameter updates involve only elementary operations (exponentiation and multiplication) on a handful of variables and a single N -length dot product. The algorithm stores one momentum value per neural network parameter (similar to momentum-based optimizers) and $2 \times N_{NHC}$ variables for the thermostat chain (e.g., 2×10 variables for the tested problems). This is comparable to algorithms like SGD with momentum or Adam (which store one or two extra tensors of size N). Therefore, memory usage scales as $O(N)$, which is of essentially the same order as standard training. Moreover, the extra computations for thermostat variable integration are a fixed small cost, independent of model size. No large auxiliary data structures are needed beyond what typical optimizers use.

In practice, training time depends not only on per-iteration cost (discussed above) but also on the number of iterations required to reach a target performance. Traditional ensemble methods like ensembled early stopping, bagging, or boosting, require an independent training run per ensemble member prior to aggregation. In contrast, since it samples near-optimal solutions rather than seeking a single loss minimum, simmering generates its entire ensemble implicitly within a single training run. This reduces the overall training time significantly compared to traditional ensemble methods, particularly when convergence is time-consuming, e.g., for large Transformers.

Model System Architecture

The examples shown in the main text demonstrate the performance of simmering on a variety of neural network problems. Different architectures, activation functions, loss functions and initializations are used to highlight the breadth of contexts in which sufficient training can be applied.

We implemented simmering using the TensorFlow library in Python. For each test case, the built-in training step in TensorFlow is replaced with the Verlet integration scheme described in Equations (1)–(9) and Algorithm 1 to iteratively produce sets of weights and biases. The acceleration $a_i(t + \Delta t/2)$ in Equation (4) (line 20 of Algorithm 1) for each iteration is supplied by the automatic differentiation of the loss with respect to the weights and biases in TensorFlow. Unless otherwise indicated, both the real particles (neural network parameters) and virtual particles had a mass $m_i = Q_k = 1 \forall i, k$.

We have published an open-source version of the simmering code, which is available at Ref.¹⁰. The published code

Algorithm 1 Simmering algorithm for a generic learning problem described by a differentiable objective function L which has parameters x , based on the NHC thermostat dynamics integration in Ref.⁷. In this integration scheme, particle positions are integrated over one half-step, i.e., half of the learning rate, at a time, and corresponding velocities are integrated forward by a full time-step at a time. The objective function parameters (neural network parameters) are treated as the positions of N one-dimensional particles which evolve over time according to Supplementary Equations 1–9. In this algorithm description, the first subscript of each variable indexes the parameters (i for the i^{th} parameter, k for the k^{th} even or odd NHC variable), and the second subscript indicates the timestep of the associated variable. Note that since the NHC particles are integrated differently based on their chain position (even or odd), the virtual particle index takes on values $k = 1, \dots, N_{NHC}/2$. Unless otherwise indicated (e.g., with a particle index of *all*), operations are performed element-wise for all elements in the given vector(s). If no initialization is specified, assume that a vector’s initial value is 0 for all elements.

Require: $T > 0$: Thermostat temperature

Require: Δt : Learning rate

Require: $L(x)$: Objective function L with parameters x

Require: $x_{all,0}$: Initial neural network parameter vector, of dimension N

Require: m : Neural network parameter mass vector, of dimension N

Require: t_{max} : Number of iterations

Require: Q : NHC particle mass vector, of dimension N_{NHC}

```

1: function POSITION_FORWARD( $a, b, dt$ )                                ▷ Integrator for parameters  $x_i$  and positions  $s_{2k}, s_{2k-1}$ .
2:   return  $a + dt \cdot b$ 
3: function VELOCITY_FORWARD( $a, b, c, dt$ )                            ▷ Integrator for velocities ( $v_i, v_{s_{2k}}, v_{s_{2k-1}}$ ).
4:   return  $a \exp(-dt \cdot c) + dt \cdot b \cdot \exp(-dt/2 \cdot c)$ 
5: function ACCELERATION_S1( $v, m, T, N, Q_1$ )                        ▷ Acceleration of the first NHC particle ( $a_{s_1}$ ).
6:   return  $(\sum_j m_j v_j^2 - N \cdot T)/Q_1$ 
7: function ACCELERATION_SK( $v, T, b, c$ )                            ▷ Acceleration of all other NHC particles ( $a_{s_{2k}}, a_{s_{2k-1} \neq 1}$ ).
8:   return  $(c \cdot v^2 - T)/b$ 
9: function REAL_T( $v, m$ )                                            ▷ For computing the parameter temperature.
10:  return  $(\sum_j m_j v_j^2)/N$ 
11:  $v_{i,0} \leftarrow \sqrt{\frac{T}{m}}$                                           ▷ Initialize velocity vector elements.
12:  $T_{real} \leftarrow T$                                               ▷ Define initial parameter temperature.
13:  $t \leftarrow 0$                                                     ▷ Initialize timestep.
14: while  $t < t_{max}$  do
15:    $x_{i,t+1/2} \leftarrow \text{POSITION\_FORWARD}(x_{i,t}, v_{i,t}, \Delta t/2)$     ▷ First half-step update to  $x_i$ .
16:    $s_{2k,t+1/2} \leftarrow \text{POSITION\_FORWARD}(s_{2k,t}, v_{s_{2k},t}, \Delta t/2)$   ▷ First half-step update to  $s_{2k}$ .
17:    $a_{s_1,t} \leftarrow \text{ACCELERATION\_S1}(v_{all,t}, m_{all}, T, N, Q_1)$ 
18:    $a_{s_{2k-1 \neq 1},t} \leftarrow \text{ACCELERATION\_SK}(v_{s_{2(k-1)},t}, T, Q_{2k-1}, Q_{2(k-1)})$ 
19:    $v_{s_{2k-1},t+1/2} \leftarrow \text{VELOCITY\_FORWARD}(v_{s_{2k-1},t}, a_{s_{2k-1},t}, v_{s_{2k},t}, \Delta t/2)$   ▷ First half-step update to  $v_{s_{2k-1}}$ .
20:    $a_{i,t+1/2} \leftarrow -\nabla_{x_i} L(x_{all,t+1/2})/m_i$                 ▷ Use loss gradient here!
21:    $v_{i,t+1} \leftarrow \text{VELOCITY\_FORWARD}(v_{i,t}, a_{i,t+1/2}, v_{s_1,t+1/2}, \Delta t)$   ▷ Full-step update to  $v_i$ .
22:    $s_{2k-1,t+1} \leftarrow \text{POSITION\_FORWARD}(s_{2k-1,t}, v_{s_{2k-1},t+1/2}, \Delta t)$   ▷ Full-step update to  $s_{2k-1}$ .
23:    $a_{s_{2k},t+1/2} \leftarrow \text{ACCELERATION\_SK}(v_{s_{2k-1},t+1/2}, T, Q_{2k}, Q_{2k-1})$ 
24:    $v_{s_{2k \neq N_c},t+1} \leftarrow \text{VELOCITY\_FORWARD}(v_{s_{2k},t}, a_{s_{2k},t+1/2}, v_{s_{2k+1},t+1/2}, \Delta t)$   ▷ Full-step update to  $v_{s_{2k}}$ .
25:    $v_{s_{N_c},t+1} \leftarrow \text{VELOCITY\_FORWARD}(v_{s_{N_c},t}, a_{s_{N_c},t+1/2}, 0, \Delta t)$   ▷ Full-step update to  $v_{s_{N_c}}$ .
26:    $x_{i,t+1} \leftarrow \text{POSITION\_FORWARD}(x_{i,t+1/2}, v_{i,t+1}, \Delta t/2)$     ▷ Last half-step update to  $x_i$ .
27:    $s_{2k,t+1} \leftarrow \text{POSITION\_FORWARD}(s_{2k,t+1/2}, v_{s_{2k},t+1}, \Delta t/2)$   ▷ Last half-step update to  $s_{2k}$ .
28:    $a_{s_1,t+1} \leftarrow \text{ACCELERATION\_S1}(v_{all,t+1}, m_{all}, T, N, Q_1)$ 
29:    $a_{s_{2k-1 \neq 1},t+1} \leftarrow \text{ACCELERATION\_SK}(v_{s_{2(k-1)},t+1}, T, Q_{2k-1 \neq 1}, Q_{2(k-1)})$ 
30:    $v_{s_{2k-1},t+1} \leftarrow \text{VELOCITY\_FORWARD}(v_{s_{2k-1},t+1/2}, a_{s_{2k-1},t+1}, v_{s_{2k},t+1}, \Delta t/2)$   ▷ Last half-step update to  $v_{s_{2k-1}}$ .
31:    $T_{real} \leftarrow \text{REAL\_T}(v_{all,t+1}, m_{all})$                 ▷ Optional: Track parameter temperature throughout sampling.
32:    $t \leftarrow t + 1$ 

```

allows for retrofitting and ab initio sufficient training to be conducted on the CIFAR-10 dataset, with sample analysis shown in Supplementary Fig. 3.

Retrofitting Overfit Networks

To retrofit a neural network, we use the final set of (overfit) weights produced by a traditional optimization algorithm and a first order approximation of the neural network parameters’ final velocities as the retrofitted neural network’s initial conditions. The final parameter velocities can be computed using the last and second-last iterations’ weights,

$$v_{i,t} \approx \frac{x_{i,t} - x_{i,t-1}}{\Delta t}, \quad (10)$$

where Δt is the learning rate. We use both the final weights and velocities to ensure that the neural network is initialized in the exact location in phase space where the optimizer stopped. We then define a temperature schedule for the thermostat, starting at $T = 0$ as the network was not coupled to a thermostat during optimization. The published code has the option to implement a temperature schedule that increases in a step-wise manner at equal intervals from $T = 0$ to the user’s target temperature choice. Main Fig. 1a shows an example of this temperature schedule, and this type of step-wise temperature change was used for all examples shown in panels Main Fig. 1e-f. Once the target temperature is reached, we collect a finite-temperature ensemble to generate ensemble predictions. For the classification examples shown in Main Fig. 1e, majority voting was used to aggregate the classifier predictions. For the regression examples in Main Fig. 1c,d,f, the ensemble samples were averaged to obtain the ensemble prediction.

In each example shown in this work, the overfit networks were produced by training with the Adam optimizer, with its default parameters in TensorFlow, and a constant learning rate of $\Delta t = 0.002$. In all cases, the batch size is taken to be the size of the entire dataset.

Three different datasets were used to produce the classification results in Fig. 1e (main text): the MNIST handwritten digits dataset¹¹, the Higgs dataset¹², and the Iris dataset¹³.

The network architecture for learning the MNIST dataset was a LeNet-5 convolutional neural network¹⁴, modified to have ReLU activations rather than sigmoid activations for all layers. Categorical cross-entropy was used as the loss function. 10,000 images were used for training, and 1000 for testing, both of which were selected randomly without overlap from the dataset. The network weights were initialized using the Glorot normal initializer in TensorFlow, and the network was trained using the Adam optimizer for 300 epochs. During simmering, the thermostat temperature was increased from $T_{\text{initial}} = 0$ to $T_{\text{target}} = 0.001$ in steps of $\Delta T = 0.001$ every 200 iterations, with a learning rate of $\Delta t = 0.002$. Simmering was conducted for 6,000 iterations, and the last 4,000 iterations’ neural network parameters were used for ensemble predictions.

For the Higgs dataset example, the network architecture consisted of 4 512-unit hidden layers, and a linear output layer. The hidden layers were given an exponential linear unit (ELU) activation. The loss, which was chosen to be binary cross-entropy, was computed from logits to account for the linear output. 10,000 samples were used for training, and 1000 samples were used for testing, selected randomly with no overlap from the Higgs dataset. The network weights were initialized using the TensorFlow Glorot uniform initializer. The network was trained using the Adam optimizer for 1000 epochs. During simmering, the thermostat temperature was increased from $T_{\text{initial}} = 0$ to $T_{\text{target}} = 0.0001$ in steps of

$\Delta T = 0.0001$ every 500 iterations, with a learning rate of $\Delta t = 0.002$. Simmering was carried out for 15,000 iterations, and the models resulting from the last 12,000 iterations of training were used for producing ensemble predictions.

For the Iris dataset example, the network architecture consisted of 3 hyperbolic tangent-activated hidden layers, and a linear output layer. The first hidden layer had 100 units, and the subsequent two hidden layers had 50 units. The categorical cross-entropy was used as the network loss. The Iris dataset consists of four input features, but we used only the “sepal width” and the “petal width” features to classify the flowers. 112 samples were used for training, and 38 were used for testing, partitioned randomly without overlap. The input features were linearly rescaled based on the training data features to map to the range $[-1, 1]$. The network weights were initialized using the TensorFlow Glorot normal initializer, and trained using the Adam optimizer for 200 epochs with no batching. During simmering, the thermostat temperature was increased from $T_{\text{initial}} = 0$ to $T_{\text{target}} = 0.1$ in steps of $\Delta T = 0.01$ every 200 iterations, with a learning rate of $\Delta t = 0.002$. The simulation was carried out for 10,000 iterations, and 10% of the last 7000 iterations were used for ensemble predictions.

Two datasets were used to produce the regression retrofitting examples shown in Fig. 1f (main text): a noisy sine curve and the Auto-MPG dataset¹⁵. For both regression datasets, “min-max” scaling was used during training, which performs a linear transformation on the input data such that all features map to the range $[-1, 1]$. The parameters of the linear transformation are set before training based on the training data feature scales. The inverse of this linear scaling is applied to the model output before it is compared with the training data targets.

The noisy sine curve data was generated by adding normally distributed noise to a sine curve in the following manner:

$$y = \sin(2\pi x) + 0.1 \mathcal{N}(\mu = 0, \sigma = 1). \quad (11)$$

The inputs used were a set of 101 equally spaced points over the interval $[-1, 1]$. The dataset was split into a training set of 65 points and a test set of 36 points, partitioned randomly with no overlap. The neural network used had two 20-unit hidden layers, and 1 output node. The two hidden layers had a hyperbolic tangent activation, and the output layer had a linear activation. The loss function used was sum-squared error (SSE). The network weights were initialized using the built-in TensorFlow Glorot normal initializer. The Adam optimizer was used to train the network weights for 2000 epochs. During simmering, the thermostat temperature was increased from $T_{\text{initial}} = 0$ to $T_{\text{target}} = 0.05$ in steps of $\Delta T = 0.01$ every 1000 iterations, with a learning rate of $\Delta t = 0.002$. The simulation was conducted for 10,000 iterations total, and the final 3,000 iterations contributed to the ensemble prediction shown in Fig. 1d (main text).

The Auto-MPG dataset was used for both single input (S) and multivariate (M) regression. In the single input case, only the “horsepower” feature was used to predict the target (miles per gallon), but in the multivariate case, all six features in the dataset to predict the same target (miles per gallon). For the single input case, the neural network architecture consisted of 2 ReLU-activated 64-unit hidden layers, and a linear output layer. The train/test split for the Auto-MPG dataset was 80/20 (313 train samples, 79 test samples) partitioned randomly with no overlap. The loss function used was SSE. The network weights were initialized using the TensorFlow Glorot normal initializer, and trained for 3500 epochs with the Adam optimizer. During simmering, the thermostat temperature was increased from $T_{\text{initial}} = 0$ to $T_{\text{target}} = 0.4$ in steps of $\Delta T = 0.1$ every 200 iterations, with a learning rate of $\Delta t = 0.001$. The simulation was conducted for 12,000 iterations total, and the final 6,000 iterations were selected for use in the ensemble prediction.

For the multivariate Auto-MPG training problem, the neural network architecture consisted of 2 hyperbolic tangent-

activated 64-unit hidden layers, and a linear output layer. The train/test split for the Auto-MPG dataset was 315 train/77 test, partitioned randomly with no overlap. The loss function used was mean-squared error (MSE). The network weights were initialized using the TensorFlow Glorot normal initializer, and trained for 1500 epochs with the Adam optimizer. During simmering, the thermostat temperature was increased from $T_{\text{initial}} = 0$ to $T_{\text{target}} = 0.5$ in steps of $\Delta T = 0.1$ every 200 iterations, with a learning rate of $\Delta t = 0.002$. The simulation was conducted for 10,000 iterations total, and the final 6,000 iterations were selected for use in the ensemble prediction.

Ab Initio Sufficient Training

For ab initio sufficient training, simmering was employed at the outset and a constant temperature was maintained for the entire duration of the training process.

To generate the classification example in Main Fig. 2, the Iris dataset was used. The choice of dataset partitioning, network architecture, batch size and loss function were the same as in the retrofitting example. The weights were initialized using a Glorot normal initialization, and simmering was employed from the outset at a thermostat temperature of $T = 0.002$ for 25,000 iterations with a learning rate of $\Delta t = 0.001$. Over the last 10,000 iterations, 2,000 model samples are randomly selected for the ensemble prediction. Simmering was conducted on models with 36 different random seeds (resulting in distinct instances of the Glorot normal initializations) while keeping the data train/test partition fixed, and the ensemble majority prediction was computed based on the total of votes across from all 36 replications’ sampled models. In Main Fig. 2a, the result of the ensemble majority vote decision boundary is shown. The background colour in Main Fig. 2a is a weighted average of the three class colours, and reflects what proportion of the ensemble of models voted for each Iris species.

For the regression example in Main Fig. 2, the Auto-MPG dataset was used. As in the single variable retrofitting example, the “horsepower” feature was used to predict the target (miles per gallon). The dataset was partitioned such that there were 300 training samples and 92 test samples, categorized randomly with no overlap. In this case, the network architecture consisted of 1 10-unit hyperbolic tangent-activated hidden layer, and a linear output. MSE was used as the loss function. The weights were initialized using a modified Glorot normal initialization. The overall distribution of weights for each layer has the same mean and width as the corresponding Glorot normal distribution, but the range $[-2\sigma, 2\sigma]$ is split into n equal segments (where n is the number of input nodes to that layer), and each weight’s value is generated from a normal distribution centred on the midpoint of one of the segments. Simmering was employed from the outset at a thermostat temperature of $T = 1$ for 40,000 iterations with a learning rate of $\Delta t = 0.002$. In Main Fig. 2b, the ensemble predictions are plotted every 40 iterations past 1000 iterations, but all 39,000 iterations are used to produce the ensemble prediction and the uncertainty distributions shown in Main Figures 2e and 2f.

Comparison With Other Overfitting Prevention Methods

We conducted two comparisons between ab initio simmering and other overfitting prevention methods: one for the CIFAR-10 dataset,¹⁶ to benchmark our algorithm’s performance on a standard large image classification problem, and the Portuguese-English translation dataset from the Open Translation Project,¹⁷ to show the capacity of simmering for natural language processing with Transformers. All experiments were simulated using TensorFlow¹⁸ using the module or function’s default

parameters unless otherwise indicated.

For training on the CIFAR-10 dataset, we used the same train-test (50k train/10k test) split and preprocessing steps as Ref.¹⁹. For both optimization and simmering we trained with a batch size of 128 images. For all cases we employed the ConvNet architecture as presented in Ref.¹⁶ and used in corresponding comparisons in Ref.¹⁹. We created a validation set by taking 5000 samples from the CIFAR-10 training set and used the performance on this subset to select the simmering thermostat parameters: $m_i = 1 \times 10^{-7} \forall i$, $Q_k = 1 \times 10^{-7} \forall k$, $T_{thermostat} = 1 \times 10^{-8}$, $\Delta t = 1 \times 10^{-5}$. For overfitting mitigation with optimization (early stopping, dropout), we used TensorFlow’s “SGD” optimizer¹⁸ as-is. For early stopping, we used TensorFlow’s “early stopping” callback with its default parameters, and for the dropout layers we used the dropout rates given in Ref.¹⁹. For all cases, the loss function was the Kullback-Leibler divergence in TensorFlow.¹⁸

The single early stopped model is provided as a reference point for the accuracy reached by a single optimized model before overfitting begins. The particular model presented was chosen by its proximity to the average accuracy of the corresponding ensemble’s members (71.60%). The ensemble of early stopped models was generated by training 100 independent models with only optimization and stopping training as soon as the validation loss began to increase. The ensemble of early stopped models had an upper limit of 50 training epochs but the sampled models trained for an average of 14.56 epochs (maximum 21 epochs) before being stopped by the callback. The ensemble predictions for both dropout and ab initio simmering were generated after 20 training epochs. The models corresponding to the last 2000 weight updates were included in the simmering ensemble. The dropout-equipped model’s inference mode prediction (in TensorFlow¹⁸) was taken to be its ensemble prediction²⁰ since it has no explicit ensemble to aggregate. The early stopping and simmering ensembles were each aggregated by majority voting.²¹ The three ensemble prediction distributions are presented in Supplementary Fig. 3 for a comparison of the ensembling methods. The images shown in Supplementary Fig. 3 were classified as relatively “easy” or “difficult” to learn based on their pixel intensity standard deviation and Shannon entropy.

For the transformer performance assessment, we employed a reduced version of the transformer architecture presented in Ref.²² with 4 decoder layers, 128-dimensional embeddings, and a 512-unit internal feedforward layer, to learn the translation dataset. The loss function was Sparse Categorical Crossentropy in TensorFlow.¹⁸ The tokenizers were pre-trained BERT tokenizers for English and for Portuguese available from Tensorflow.¹⁸ To implement the transformer, we used TensorFlow’s built-in base functions, early stopping callback and dropout layers. For the optimization trials, we used the Adam optimizer with a constant learning rate of $\Delta t = 1 \times 10^{-5}$, as larger learning rates led to slow or no convergence. All training experiments had a batch size of 64. The early stopped model ensemble consists of 10 independently trained models and had an average training time of 53.1 epochs (maximum 72 epochs). The single early stopping model was chosen for presentation by its proximity to the ensemble member test accuracy average (39.17%). Based on the average training time of the early stopped model ensemble, we optimized a model with dropout for 60 epochs. We used the standard training/validation/test split of the translation dataset and used the validation data performance to choose simmering thermostat parameters, $m_i = 1 \times 10^{-8} \forall i$, $Q_k = 1 \times 10^{-8} \forall k$, $T_{thermostat} = 1 \times 10^{-7}$, and employed the same learning rate for simmering as for optimization of $\Delta t = 1 \times 10^{-5}$. We found that simmering outperformed the accuracy of dropout and early stopping after only 21 epochs, and reported the last epoch’s parameter sets’ ensemble performance in Figure 3. For all training scenarios, we used a teacher-forced inference method for accuracy evaluation efficiency. The ensemble predictions were all aggregated in the same way as their corresponding ensembles in the CIFAR-10 performance comparison.

Supplementary References

1. Li, H., Xu, Z., Taylor, G., Studer, C. & Goldstein, T. *Visualizing the Loss Landscape of Neural Nets* in *Advances in Neural Information Processing Systems* **31** (2018).
2. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2017. arXiv: 1412.6980 [cs.LG].
3. Hinton, G., Vinyals, O. & Dean, J. *Distilling the Knowledge in a Neural Network* Mar. 2015. arXiv: 1503.02531 [stat]. (2025).
4. Wenzel, F. *et al. How Good Is the Bayes Posterior in Deep Neural Networks Really?* July 2020. arXiv: 2002.02405 [stat]. (2025).
5. Kingma, D. P. & Welling, M. *Auto-Encoding Variational Bayes* Dec. 2022. arXiv: 1312.6114 [stat]. (2025).
6. Ding, N. *et al. Bayesian Sampling Using Stochastic Gradient Thermostats* in *Advances in Neural Information Processing Systems* **27** (Curran Associates, Inc., 2014). (2025).
7. Jang, S. & Voth, G. A. Simple Reversible Molecular Dynamics Algorithms for Nosé–Hoover Chain Dynamics. *The Journal of Chemical Physics* **107**, 9514–9526. issn: 0021-9606, 1089-7690. <https://pubs.aip.org/jcp/article/107/22/9514/478184/Simple-reversible-molecular-dynamics-algorithms> (2024) (Dec. 8, 1997).
8. Yamamoto, T. Comment on “Comment on ‘Simple Reversible Molecular Dynamics Algorithms for Nosé–Hoover Chain Dynamics’ ” [J. Chem. Phys. 110, 3623 (1999)]. *J Chem Phys* **124**, 217101. issn: 0021-9606, 1089-7690. <https://pubs.aip.org/jcp/article/124/21/217101/895812/Comment-on-Comment-on-Simple-reversible-molecular> (2025) (June 7, 2006).
9. Martyna, G. J., Klein, M. L. & Tuckerman, M. Nosé–Hoover Chains: The Canonical Ensemble via Continuous Dynamics. *J Chem Phys* **97**, 2635–2643 (1992).
10. Babayan, I., Aliahmadi, H. & van Anders, G. *Simmering: Sufficient Training of Neural Networks in Python* Zenodo, Aug. 13, 2025. <https://zenodo.org/doi/10.5281/zenodo.13993441> (2025).
11. Deng, L. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine* **29**, 141–142 (2012).
12. Whiteson, D. *HIGGS* UCI Machine Learning Repository. 2014.
13. Fisher, R. A. *Iris* UCI Machine Learning Repository. 1936.
14. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* **86**, 2278–2324 (1998).
15. Quinlan, R. *Auto MPG* UCI Machine Learning Repository. 1993.
16. Krizhevsky, A. & Hinton, G. *Learning Multiple Layers of Features from Tiny Images* Technical Report (University of Toronto, Toronto, ON, Canada, Apr. 8, 2009). <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>.
17. Ye, Q., Devandra, S., Matthieu, F., Sarguna, P. & Graham, N. *When and Why Are Pre-Trained Word Embeddings Useful for Neural Machine Translation* in *Human Language Technologies* The Nations of the Americas Chapter of the Association for Computational Linguistics (New Orleans, Louisiana, United States, 2018).

18. Martín Abadi *et al.* *TensorFlow: Large-scale Machine Learning on Heterogeneous Systems* 2015. <https://www.tensorflow.org/>.
19. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* **15**, 1929–1958. <http://jmlr.org/papers/v15/srivastava14a.html> (2014).
20. Hara, K., Saitoh, D. & Shouno, H. in *Artificial Neural Networks and Machine Learning – ICANN 2016* (eds Villa, A. E., Masulli, P. & Pons Rivero, A. J.) 72–79 (Springer International Publishing, Cham, 2016). ISBN: 978-3-319-44780-3 978-3-319-44781-0. http://link.springer.com/10.1007/978-3-319-44781-0_9 (2025).
21. Dietterich, T. G. *Ensemble Methods in Machine Learning in International Workshop on Multiple Classifier Systems* (Springer, 2000), 1–15.
22. Vaswani, A. *et al.* *Attention Is All You Need* in *Advances in Neural Information Processing Systems* (eds Guyon, I. *et al.*) **30** (Curran Associates, Inc., 2017). https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.