



HAL
open science

PTFlash : A vectorized and parallel deep learning framework for two-phase flash calculation

Jingang Qu, Thibault Faney, Jean-Charles de Hemptinne, Soleiman Yousef,
Patrick Gallinari

► **To cite this version:**

Jingang Qu, Thibault Faney, Jean-Charles de Hemptinne, Soleiman Yousef, Patrick Gallinari. PT-Flash : A vectorized and parallel deep learning framework for two-phase flash calculation. *Fuel*, 2023, 331, Part 1, pp.125603. 10.1016/j.fuel.2022.125603 . hal-03659647v3

HAL Id: hal-03659647

<https://hal.science/hal-03659647v3>

Submitted on 10 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0
International License

PTFlash : A vectorized and parallel deep learning framework for two-phase flash calculation*

Jingang Qu^{a,b,*}, Thibault Faney^b, Jean-Charles de Hemptinne^b, Soleiman Yousef^b, Patrick Gallinari^{a,c}

^a*Sorbonne Université, CNRS, ISIR, F-75005 Paris, France*

^b*IFPEN, France*

^c*Criteo AI Lab, Paris, France*

Abstract

Phase equilibrium calculations are an essential part of numerical simulations of multi-component multi-phase flow in porous media, accounting for the largest share of the computational time. In this work, we introduce a fast and parallel framework, *PTFlash*, that vectorizes algorithms required for two-phase flash calculation using PyTorch, and can facilitate a wide range of downstream applications. Vectorization promotes parallelism and consequently leads to attractive hardware-agnostic acceleration. In addition, to further accelerate *PTFlash*, we design two task-specific neural networks, one for predicting the stability of given mixtures and the other for providing estimates of the distribution coefficients, which are trained offline and help shorten computation time by sidestepping stability analysis and reducing the number of iterations to reach convergence.

The evaluation of *PTFlash* was conducted on three case studies involving hydrocarbons, CO_2 and N_2 , for which the phase equilibrium was tested over a large range of temperature, pressure and composition conditions, using the Soave-Redlich-Kwong (SRK) equation of state. We compare *PTFlash* with an

*This is the manuscript accepted by Fuel journal. For the formal publication, please refer to <https://doi.org/10.1016/j.fuel.2022.125603>.

**The share link <https://authors.elsevier.com/a/1fj0m3iH4IIcE> allows free access to the article for 50 days from 7 Sep 2022.

*Corresponding author

Email address: jingang.qu@sorbonne-universite.fr (Jingang Qu)

in-house thermodynamic library, *Carnot*, written in C++ and performing flash calculations one by one on CPU. Results show speed-ups of up to two order of magnitude on large scale calculations, while maintaining perfect precision with the reference solution provided by *Carnot*.

Keywords: Flash calculation, Two-phase equilibrium, Vectorization, Deep learning

1. Introduction

Numerical simulation of multi-component multi-phase flow in porous media is an essential tool for many subsurface applications, from reservoir simulation to long term CO_2 storage. A core element of the simulator for such applications is to determine the phase distribution of a given fluid mixture at equilibrium, also known as flash calculation. Starting with the seminal work of Michelsen [1, 2], researchers have developed robust and efficient algorithms for isothermal two-phase flash calculation. These algorithms have been implemented in the IFPEN thermodynamic C++ library *Carnot*.

Nonetheless, flash calculations still account for the majority of simulation time in a large range of subsurface applications [3, 4]. In most simulators, flash calculations are performed for each grid cell at each time step. Moreover, since modern simulators tend to require higher and higher grid resolutions up to billions of grid cells [5], the share of computing time due to flash calculations is expected to increase as well. In this context, speeding up flash calculations has drawn increasing research interest.

Some efforts have been made to accelerate flash calculations. [6–8] proposed a reduction method aiming to reduce the number of independent variables by leveraging the sparsity of the binary interaction parameter matrix, resulting in a limited speed-up [4]. [9] introduced the shadow region method using the results of previous time steps to initiate the current one, which assumes that the changes in pressure, temperature, and composition of a given block are small between two adjacent time steps in typical compositional reservoir simulation.

24 [10] presented tie-line based methods, which approximate the results of flash
25 calculations through linear interpolation between existing tie-lines and can be
26 seen as a kind of look-up table. In [11–16], the authors focused on the use of
27 machine learning, which provides a collection of techniques that can effectively
28 discover patterns and regularities in data. They used support vector machine
29 [17], relevance vector machine [18] and neural networks [19] to directly predict
30 equilibrium phases and provide more accurate initial estimates for flash calcula-
31 tions. In [5, 20], researchers focused on developing faster parallel linear solvers,
32 with [5] mentioning specifically that the vectorization of partial equation of state
33 (EOS) related operations would lead to faster execution.

34 In this work, we introduce *PTFlash*, a framework for two-phase flash calcu-
35 lation based on the SRK equation of state [21]. *PTFlash* is built on the deep
36 learning framework PyTorch [22] and consists in two main elements, namely
37 the vectorization of algorithms and the use of neural networks. First, we per-
38 form a complete rewrite of two-phase flash calculation algorithms of *Carnot*
39 using PyTorch. This enables the systematic vectorization of the complex iter-
40 ative algorithms implemented in *Carnot*, allowing in turn to efficiently harness
41 modern hardware with the help of, e.g., Advanced Vector Extensions AVX for
42 Intel CPUs [23] and CUDA for Nvidia GPUs [24]. Note that vectorization of
43 complex iterative algorithms with branching is not straightforward and needs
44 specific care. Second, we replace repetitive and time consuming parts of the
45 original algorithms with deep neural networks trained on the exact solution.
46 More specifically, one neural network is used to predict the stability of given
47 mixtures, and the other is used to provide initial estimates for the iterative al-
48 gorithms. Once well trained, neural networks are seamlessly incorporated into
49 *PTFlash*. These two elements allow *PTFlash* to provide substantial speed-ups
50 compared to *Carnot*, especially so in the context of flow simulations where par-
51 allel executions of flash calculations for up to a billion grid cells are needed.

52 The rest of this article is organized as follows. In Section 2, we introduce
53 the fundamentals of isothermal two-phase flash calculation and present three
54 case studies. In Section 3, we explain how to efficiently vectorize flash calcu-

55 lation using PyTorch. In Section 4, we present two neural networks to speed
 56 up calculations. In Section 5, we demonstrate the attractive speed-up due to
 57 vectorization and the introduction of neural networks. Finally, we summarize
 58 our work and suggest future research in Section 6.

59 2. Isothermal two-phase flash calculation

60 In this section, we introduce the essential concepts of isothermal two-phase
 61 flash calculation. In the following, without loss of generality, we consider the
 62 equilibrium between the liquid and vapor phases.

63 2.1. Problem setting

64 We consider a mixture of N_c components. Given pressure (P), temperature
 65 (T) and feed composition ($\mathbf{z} = (z_1, \dots, z_{N_c})$), the objective of flash calculation
 66 is to determine the system state at equilibrium: single phase or coexistence
 67 of two phases. In the latter case, we need to additionally compute the molar
 68 fraction of vapor phase θ_V , the composition of the liquid phase \mathbf{x} and that of the
 69 vapor phase \mathbf{y} . These properties are constrained by the following mass balance
 70 equations:

$$x_i(1 - \theta_V) + y_i\theta_V = z_i, \quad \text{for } i = 1, \dots, N_c \quad (1a)$$

$$\sum_{i=1}^{N_c} x_i = \sum_{i=1}^{N_c} y_i = 1 \quad (1b)$$

71 In addition, the following equilibrium condition should be satisfied:

$$\frac{\varphi_i^L(P, T, \mathbf{x})}{\varphi_i^V(P, T, \mathbf{y})} = \frac{y_i}{x_i} \quad (2)$$

72 where the superscripts L and V refer to the liquid and vapor phases, respectively,
 73 and φ_i is the fugacity coefficient of component i , which is a known nonlinear
 74 function of P , T and the corresponding phase composition. This function de-
 75 pends on an equation of state that relates pressure, temperature and volume. In
 76 this work, we use the SRK equation of state [21] and solve it using an iterative

77 approach [25] rather than the analytical solution of the cubic equation, e.g., the
 78 Cardano’s formula, which may be subject to numerical errors in certain edge
 79 cases [26]. For more details, see Appendix A.

80 2.2. Numerical solver

81 Equations 1 and 2 form a non-linear system, which is generally solved in
 82 a two-stage procedure. First, we establish the stability of a given mixture via
 83 stability analysis (Section 2.2.1). If the mixture is stable, only one phase exists
 84 at equilibrium. Otherwise, two phases coexist. Second, we determine θ_V , \mathbf{x} and
 85 \mathbf{y} at equilibrium through phase split calculations (Section 2.2.2).

86 2.2.1. Stability analysis

87 A mixture of composition \mathbf{z} is stable at specified P and T if and only if its
 88 total Gibbs energy is at the global minimum, which can be verified through the
 89 reduced tangent plane distance [1]:

$$tpd(\mathbf{w}) = \sum_{i=1}^{N_c} w_i (\ln w_i + \ln \varphi_i(\mathbf{w}) - \ln z_i - \ln \varphi_i(\mathbf{z})) \quad (3)$$

90 where \mathbf{w} is a trial phase composition. If $tpd(\mathbf{w})$ is non-negative for any \mathbf{w} , the
 91 mixture is stable. This involves a constrained minimization problem, which is
 92 generally reframed as an unconstrained one:

$$tm(\mathbf{W}) = \sum_{i=1}^{N_c} W_i (\ln W_i + \ln \varphi_i(\mathbf{W}) - \ln z_i - \ln \varphi_i(\mathbf{z}) - 1) \quad (4)$$

93 where tm is the modified tangent plane distance and \mathbf{W} is mole numbers. To
 94 locate the minima of tm , we first use the successive substitution method accel-
 95 erated by the Dominant Eigenvalue Method (DEM) [27], which iterates:

$$\ln W_i^{(k+1)} = \ln z_i + \ln \varphi_i(\mathbf{z}) - \ln \varphi_i(\mathbf{W}^{(k)}) \quad (5)$$

96 It is customary to initiate the minimization with two sets of estimates, that is,
 97 vapor-like estimate $W_i = K_i z_i$ and liquid-like estimate $W_i = z_i / K_i$, where K_i

98 is the distribution coefficients, defined as y_i/x_i and initialized via the Wilson
 99 approximation [21], as follows:

$$\ln K_i = \ln \left(\frac{P_{c,i}}{P} \right) + 5.373(1 + \omega_i) \left(1 - \frac{T_{c,i}}{T} \right) \quad (6)$$

100 where $T_{c,i}$ and $P_{c,i}$ refer to the critical temperature and pressure of component
 101 i , respectively, and ω_i is the acentric factor.

102 Once converging to a stationary point (i.e., $\max(|\partial tm/\partial \mathbf{W}|) < 1.0e-6$) or
 103 a negative tm is found, successive substitution stops. If this does not happen
 104 after a fixed number of iterations (9 in our work), especially in the vicinity
 105 of critical points, we resort to a second-order optimization technique, i.e., the
 106 trust-region method [28], to minimize $tm(\mathbf{W})$, which we describe in Appendix
 107 B.1. In addition, based on the results of stability analysis, we can re-estimate
 108 K_i more accurately as z_i/W_i^L if $tm^L < tm^V$ or W_i^V/z_i otherwise, where the
 109 superscripts V and L denote the results obtained using the vapor-like and liquid-
 110 like estimates, respectively.

111 2.2.2. Phase split calculations

112 Substituting $K_i = y_i/x_i$ into Equation 1 yields the following Rachford-Rice
 113 equation [29]:

$$f_{RR}(\theta_V, \mathbf{K}) = \sum_{i=1}^{N_c} \frac{(K_i - 1)z_i}{1 + (K_i - 1)\theta_V} = 0 \quad (7)$$

114 Given $\mathbf{K} = (K_1, \dots, K_{N_c})$, we solve the above equation using the method pro-
 115 posed by [30] to get θ_V , which is detailed in Appendix C.1.

116 To obtain θ_V , \mathbf{x} and \mathbf{y} at equilibrium, phase split calculations start with
 117 the accelerated successive substitution method, as illustrated in Figure 1, and
 118 the corresponding convergence criterion is $\max(|K_i^{(k+1)}/K_i^{(k)} - 1|) < 1.0e-8$. If
 119 successive substitution fails to converge after a few iterations (9 in our work),
 120 we use the trust-region method to minimize the reduced Gibbs energy:

$$G = \sum_{i=1}^{N_c} n_i^L (\ln x_i + \ln \varphi_i^L) + \sum_{i=1}^{N_c} n_i^V (\ln y_i + \ln \varphi_i^V) \quad (8)$$

121 where $n_i^L = x_i(1 - \theta_V)$ and $n_i^V = y_i\theta_V$ are the mole numbers of liquid and vapor
 122 phases, respectively. The convergence criterion is $\max(|\partial G/\partial n_i^V|) < 1.0e-8$. For
 more details, see Appendix B.2.

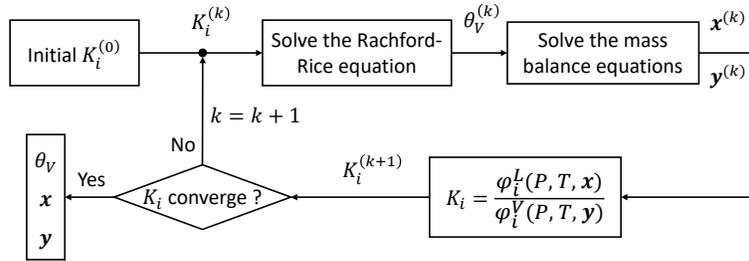


Figure 1: Successive substitution of phase split calculations

123

124 2.2.3. Strategy for two-phase flash calculation

125 We basically adopt the rules of thumb proposed by Michelsen in the book
 126 [31] to implement two-phase flash calculation, as shown in Figure 2. In the
 127 flowchart, we first initialize the distribution coefficients K_i using the Wilson
 128 approximation. Subsequently, in order to avoid computationally expensive sta-
 129 bility analysis, we carry out the successive substitution of phase split calculations
 130 3 times, which will end up with 3 possible cases: (1) θ_V is out of bounds (0,
 131 1) during iterations. (2) None of ΔG , $tpd(\mathbf{x})$ and $tpd(\mathbf{y})$ are negative, where
 132 $tpd(\mathbf{x})$ and $tpd(\mathbf{y})$ are reduced tangent plane distances using current vapor and
 133 liquid phases as trial phases, and $\Delta G = \theta_V \times tpd(\mathbf{x}) + (1 - \theta_V) \times tpd(\mathbf{y})$. (3)
 134 Any of ΔG , $tpd(\mathbf{x})$ and $tpd(\mathbf{y})$ is negative.

135 For the first two cases, we cannot be sure of the stability of the given mixture,
 136 thus continuing with stability analysis. For the third case, we can conclude that
 137 the given mixture is unstable, thereby sidestepping stability analysis. Finally,
 138 if two phases coexist, we perform phase split calculations to get θ_V , \mathbf{x} and \mathbf{y} at
 139 equilibrium.

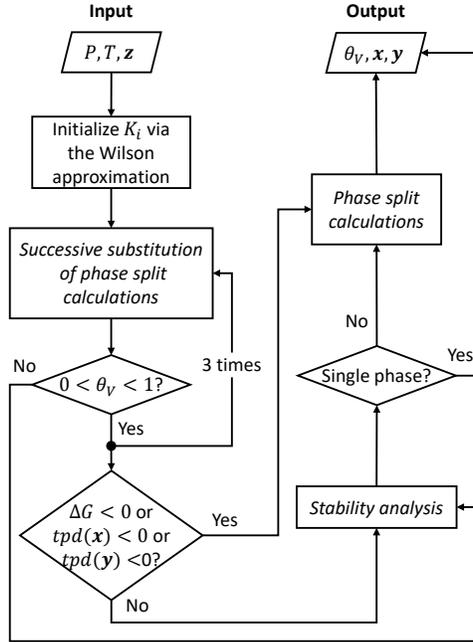


Figure 2: Flowchart of two-phase flash calculation

140 *2.3. Case studies*

141 Here, we introduce three case studies involving hydrocarbons, CO_2 and N_2 ,
 142 whose properties are shown in Table 1. In this work, we only consider the binary
 143 interaction parameter (BIP) between CH_4 and CO_2 , which is 0.0882. The BIPs
 144 between the others are 0. The first case study focuses on a system of two
 145 components (CH_4 and C_6H_{14}), and the second one involves four components
 146 (CH_4 , C_2H_6 , C_3H_8 and C_4H_{10}). For these two case studies, the ranges of
 147 pressure and temperature are 0.1MPa - 10MPa and 200K - 500K, respectively,
 148 and we consider the entire compositional space, i.e., $0 < z_i < 1$ for $i = 1, \dots, N_c$.
 149 The third case study includes all 9 components in Table 1. The bounds of
 150 pressure and temperature are 5MPa - 25MPa and 200K - 600K, respectively. In
 151 addition, from a practical perspective, given that some mixtures do not exist in
 152 nature, rather than considering the entire compositional space, we specify four
 153 different compositional ranges, as shown in Table 2, each of which represents one

Table 1: Component properties

	P_c (MPa)	T_c (K)	ω
CH_4	4.6	190.55	0.0111
C_2H_6	4.875	305.43	0.097
C_3H_8	4.268	369.82	0.1536
$n-C_4H_{10}$	3.796	425.16	0.2008
$n-C_5H_{12}$	3.3332	467.15	0.2635
C_6H_{14}	2.9688	507.4	0.296
$C_7H_{16}^+$	2.622	604.5	0.3565
CO_2	7.382	304.19	0.225
N_2	3.3944	126.25	0.039

154 of the common reservoir fluid types, namely wet gas, gas condensate, volatile
 155 oil, and black oil. Figure 3(a) shows phase diagrams of four typical reservoir
 156 fluids at fixed compositions, as defined in Appendix D, and we can see that the
 157 more heavy hydrocarbons there are, the lower the pressure range of the phase
 envelope and the less volatile the fluid is.

Table 2: Four fluid types characterized by different compositional ranges

	Wet gas	Gas condensate	Volatile oil	Black oil
CH_4	80% - 100%	60% - 80%	50% - 70%	20% - 40%
C_2H_6	2% - 7%	5% - 10 %	6% - 10%	3% - 6 %
C_3H_8	$\leq 3\%$	$\leq 4\%$	$\leq 4.5\%$	$\leq 1.5\%$
$n-C_4H_{10}$	$\leq 2\%$	$\leq 3\%$	$\leq 3\%$	$\leq 1.5\%$
$n-C_5H_{12}$	$\leq 2\%$	$\leq 2\%$	$\leq 2\%$	$\leq 1\%$
C_6H_{14}	$\leq 2\%$	$\leq 2\%$	$\leq 2\%$	$\leq 2\%$
$C_7H_{16}^+$	$\leq 1\%$	5% - 10 %	10% - 30%	45% - 65%
CO_2	$\leq 2\%$	$\leq 3.5\%$	$\leq 2\%$	$\leq 0.1\%$
N_2	$\leq 0.5\%$	$\leq 0.5\%$	$\leq 0.5\%$	$\leq 0.5\%$

158

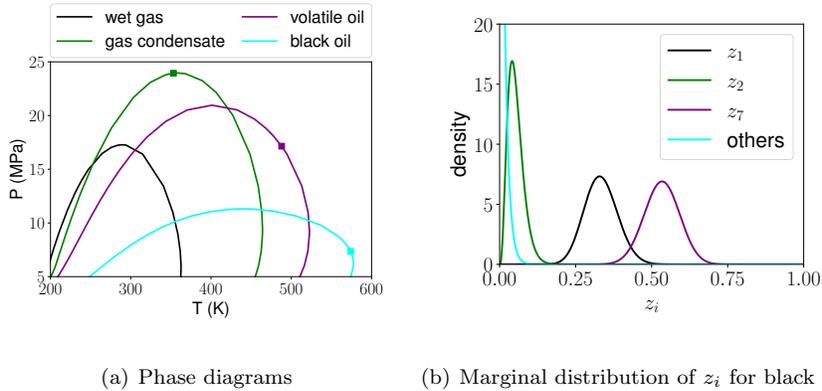


Figure 3: In Figure (a), the squares on the phase envelopes represent critical points. In Figure (b), z_1 , z_2 and z_7 are the molar fractions of CH_4 , C_2H_6 and $C_7H_{16}^+$, respectively.

159 *2.4. Data generation*

160 To efficiently sample input data including P , T and \mathbf{z} , we first use Latin
 161 Hypercube Sampling (LHS) technique to take space-filling samples [32]. Subse-
 162 quently, for P and T , we linearly transform the uniform distribution $\mathcal{U}(0, 1)$ to
 163 the expected ranges. For \mathbf{z} subject to $\sum z_i = 1$, we transform a set of $\mathcal{U}(0, 1)$
 164 into the Dirichlet distribution $Dir(\boldsymbol{\alpha})$ whose support is a simplex, as follows:

$$x_i \stackrel{i.i.d.}{\sim} \mathcal{U}(0, 1) \text{ using LHS} \quad (9a)$$

$$y_i = \Gamma(\alpha_i, 1).ppf(x_i) \quad (9b)$$

$$z_i = \frac{y_i}{\sum_{i=1}^{N_c} y_i} \quad (9c)$$

165 where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_{N_c})$ is the concentration parameters of the Dirichlet dis-
 166 tribution and controls its mode, $\Gamma(\alpha_i, 1)$ is the Gamma distribution, ppf rep-
 167 represents the percent-point function, also known as the quantile function, and
 168 $\mathbf{z} = (z_1, \dots, z_{N_c}) \sim Dir(\boldsymbol{\alpha})$.

169 For the first two case studies, the concentration parameters are $\boldsymbol{\alpha} = \mathbf{1}$, i.e.,
 170 all-ones vector. For the third case study, we adjust $\boldsymbol{\alpha}$ for different fluid types to
 171 make the probability of each compositional range as large as possible, as shown

172 in Table 3. Figure 3(b) presents the marginal distribution of z_i for black oil.
 173 In summary, we sample \mathbf{z} using Equation 9 with different α specified in Table
 174 3, and then we single out the acceptable samples located in the compositional
 175 ranges defined in Table 2. In the following, unless otherwise specified, four fluid
 types are always equally represented.

Table 3: Concentration parameters α for different fluid types in Table 2

	α_1 for CH_4	α_2 for C_2H_6	α_7 for $C_7H_{16}^+$	α_i for others
Wet gas	100	5	1	1
Gas condensate	40	5	5	1
Volatile oil	55	8	20	1
Black oil	25	4	40	1

176

177 Eventually, the samples of P , T and \mathbf{z} are concatenated together to form
 178 the complete input data.

179 3. Vectorization of two-phase flash calculation

180 We vectorize the two-phase flash so that it takes as inputs $\mathbf{P} = (P_1, \dots, P_n)$,
 181 $\mathbf{T} = (T_1, \dots, T_n)$ and $\mathbf{z} = (z_1, \dots, z_n)$, where \mathbf{P} and \mathbf{T} are vectors, \mathbf{z} is a
 182 matrix, and n denotes the number of samples processed concurrently and is
 183 often referred to as the batch dimension.

184 In recent years, Automatic Vectorization (AV) has emerged and developed ¹,
 185 e.g., JAX [33], which can automatically vectorize a function through the batch-
 186 ing transformation that adds a batch dimension to its input. In this way, the
 187 vectorized function can process a batch of inputs simultaneously rather than
 188 processing them one by one in a loop. However, AV comes at the expense of
 189 performance to some extent and is slower than well-designed manual vectoriza-
 190 tion, which vectorizes a function by carefully revamping its internal operations

¹At the time of writing, PyTorch team released a fledgling library, *functorch*, which takes inspiration from JAX and supports Automatic Vectorization.

191 to accommodate to a batch of inputs. For example, matrix-vector products
 192 for a batch of vectors can be directly replaced with a matrix-matrix product.
 193 In addition, flash calculation has an iterative nature and complicated control
 194 flow, which is likely to result in the failure of AV. Consequently, for finer-grained
 195 control, more flexibility, and better performance, we manually vectorize all algo-
 196 rithms involved in flash calculation, including the solution of the SRK equation
 197 of state and the Rachford-Rice equation, stability analysis and phase split cal-
 198 culations.

199 To achieve efficient vectorization, one difficulty is asynchronous convergence,
 200 that is, for each algorithm, the number of iterations required to reach conver-
 201 gence generally varies for different samples, which hinders vectorization and
 202 parallelism. To alleviate this problem, we design a general-purpose paradigm,
 203 *synchronizer*, to save converged results in time at the end of each iteration and
 204 then remove the corresponding samples in order not to waste computational
 205 resources on them in the following iterations, which is achieved by leveraging
 206 a one-dimensional Boolean mask encapsulating convergence information to effi-
 207 ciently access data in vectors and matrices, as follows:

$$\mathbf{X}^{(k+1)} \leftarrow f(\mathbf{X}^{(k)}) \quad (10a)$$

$$\text{Save } \mathbf{X}^{(k+1)}[\text{mask}] \text{ to } \widetilde{\mathbf{X}} \quad (10b)$$

$$\mathbf{X}^{(k+1)} \leftarrow \mathbf{X}^{(k+1)}[\sim \text{mask}] \quad (10c)$$

$$k \leftarrow k + 1 \quad (10d)$$

208 where k is the number of iterations, $f(\mathbf{X})$ is a vectorized iterated function tak-
 209 ing as input $\mathbf{X} \in \mathbb{R}^{n \times m}$ (n is the batch dimension, i.e., number of samples, and
 210 m is the dimension of X), $\widetilde{\mathbf{X}}$ is a placeholder matrix used to save converged
 211 results, mask is a Boolean vector where True means convergence, and \sim de-
 212 notes the logical NOT operator. The number of unconverged samples gradually
 213 decreases as a result of incremental convergence. For the full version of *synchro-*
 214 *nizer*, refer to Appendix E.1. We can use *synchronizer* to wrap and vectorize

215 any iterative algorithm. For instance, we illustrate how to perform vectorized
216 stability analysis in Appendix E.2.

217 The efficiency of *synchronizer* may be questioned because previously con-
218 verged samples are still waiting for unconverged ones before moving to the next
219 step. This is true, but the situation is not as pessimistic since we try to shorten
220 the waiting time as much as possible. For example, if successive substitution
221 fails to converge quickly, we immediately use the trust-region method. In any
222 case, the delay caused by waiting is insignificant compared to the acceleration
223 due to vectorization. Furthermore, we leverage neural networks to provide more
224 accurate initial estimate $\mathbf{X}^{(0)}$ so that all samples converge as simultaneously as
225 possible, thereby reducing asynchrony, which we will present in Section 4.

226 Once all algorithms are well vectorized, another problem is how to globally
227 coordinate different subroutines. To this end, we add barrier synchronization
228 to the entry points of stability analysis and phase split calculations in Figure
229 2, which can avoid any subroutine connected to it proceeding further until all
230 others terminate and arrive at this barrier.

231 We also optimized the code using TorchScript [22], allowing for more efficient
232 execution through algebraic peephole optimizations and fusion of some opera-
233 tions, and more practical asynchronous parallelism without the Python global
234 interpreter lock [34], whereby vapor-like and liquid-like estimates are dealt with
235 in parallel in stability analysis.

236 4. Acceleration of flash calculation using neural networks

237 To further accelerate flash calculation, we create and train two task-specific
238 neural networks, classifier and initializer. The classifier is used to predict the
239 probability p that a given mixture is stable, i.e., $p = \text{classifier}(P, T, \mathbf{z})$, which
240 involves a binary classification problem. It can predict the stability of most
241 samples, thereby bypassing stability analysis and saving time. The initializer
242 is able to initialize K_i more accurately than the Wilson approximation, i.e.,
243 $\ln K_i = \text{initializer}(P, T, \mathbf{z})$, which relates to a regression problem. It can reduce

244 the number of iterations required to reach convergence and alleviate the asyn-
 245 chronous convergence we introduced before. Note that the hyper-parameters
 246 of neural networks presented below, e.g., the number of units and layers, are
 247 dedicated to the case study containing 9 components. Nonetheless, the basic
 248 architecture of neural networks and the training methods can be generalized to
 249 any case.

250 4.1. Classifier

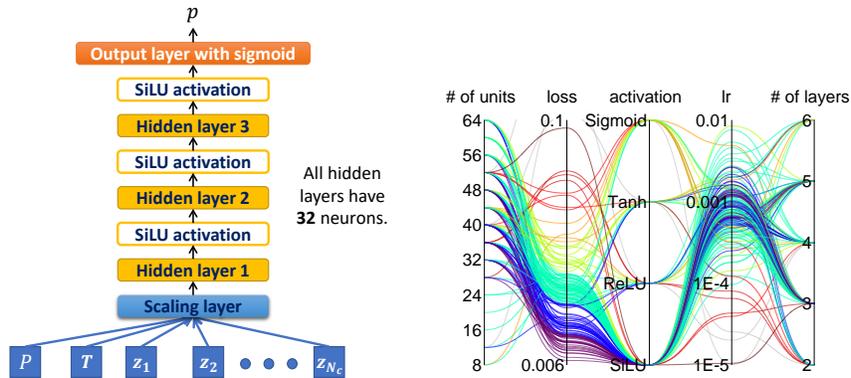
251 4.1.1. Architecture

252 As shown in Figure 4(a), the classifier has 3 hidden layers with 32 neurons
 253 and using the SiLU activation function [35–37]. The output layer has only one
 254 neuron and uses the sigmoid activation function compressing a real number to
 255 the range (0, 1). The input \mathbf{x} consists of P , T and \mathbf{z} , and the output is the
 256 probability p that a given mixture is stable. The scaling layer standardizes the
 257 inputs as $(\mathbf{x} - \mathbf{u})/\mathbf{s}$, where \mathbf{u} and \mathbf{s} are the mean and standard deviation of \mathbf{x}
 258 over the training set. To train the classifier, we use the binary cross-entropy
 259 (bce), which is the de-facto loss function for binary classification problems and
 260 defined as:

$$\text{bce}(y, p) = y \ln p + (1 - y) \ln(1 - p) \quad (11)$$

261 where y is either 0 for unstable mixtures or 1 for stable ones.

262 The architecture of the classifier is obtained by tuning hyper-parameters us-
 263 ing Tree-Structured Parzen Estimator optimization algorithm [38] with Asyn-
 264 chronous Successive Halving algorithm [39] as an auxiliary tool to early stop
 265 less promising trials. We create a dataset containing 100,000 samples (80%
 266 for training and 20% for validation), and then tune the hyper-parameters of the
 267 classifier with 150 trials to minimize the loss on the validation set (we use Adam
 268 [40] as optimizer and the batch size is 512), as shown in Figure 4(b). We can
 269 see that SiLU largely outperforms other activation functions.



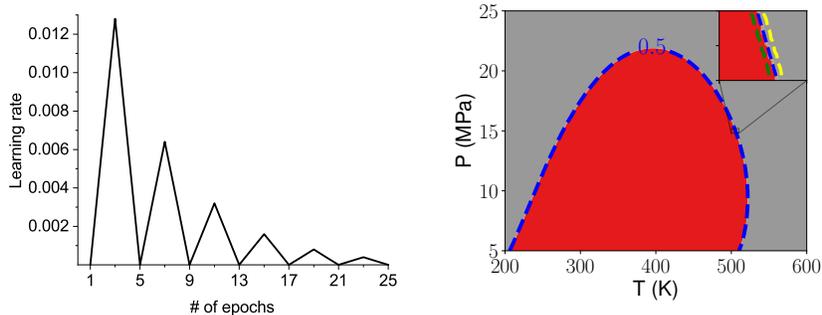
(a) The architecture of classifier for the 9-component mixture (b) Tuning hyper-parameters of classifier

Figure 4: Figure (a) shows the architecture of the classifier. Figure (b) is a parallel coordinates plot used to visualize the results of tuning hyper-parameters of the classifier, where lr stands for learning rate. The colors of lines are mapped to the value of the loss.

270 4.1.2. Training

271 We first generate one million samples in the way described in Section 2.3,
 272 and then feed them to *PTFlash* to determine stability (no need for phase split
 273 calculations), which takes about 2 seconds. Subsequently, these samples are
 274 divided into the training (70%), validation (15%) and test (15%) sets. To train
 275 the classifier, we set the batch size to 512 and use Adam with Triangular Cyclic
 276 Learning Rate (CLR) [41, 42], which periodically increases and decreases the
 277 learning rate during training, as shown in Figure 5(a). [43] claimed that CLR
 278 helps to escape local minima and has the opportunity to achieve superb perfor-
 279 mance using fewer epochs and less time. We found that Adam with and without
 280 CLR achieved similar performance, but the former converged five times faster
 281 than the latter. Early stopping is also used to avoid overfitting [44]. The total
 282 training time is about 5 minutes using Nvidia RTX 3080. The final performance
 283 of the classifier on the test set is $bce = 0.002$ and $accuracy = 99.93\%$. For a
 284 more intuitive understanding of performance, Figure 5(b) shows the contours
 285 of probabilities predicted by the classifier, where the blue contour of $p = 0.5$

286 basically coincides with the phase envelope. In the zoomed inset, the additional
 287 green and yellow contours correspond to $p=0.02$ and 0.98 , respectively.



(a) Cyclic learning rate of the classifier (b) Prediction of the classifier for volatile oil

Figure 5: Figure (a) shows how the learning rate varies cyclically. Figure (b) illustrates the contours of probabilities predicted by the classifier for volatile oil at fixed composition. The red and gray correspond to the two-phase and monophasic regions, respectively.

288 4.2. *Initializer*

289 4.2.1. *Architecture*

290 The input of the initializer includes P , T and \mathbf{z} , and its output is $\ln K_i$.
 291 The initializer has 1 hidden layer and 3 residual blocks, as shown in Figure
 292 6. Each residual block has 2 hidden layers and a shortcut connection adding
 293 the input of the first hidden layer to the output of the second [45]. All hidden
 294 layers have 64 neurons and use the SiLU activation function. The output layer
 295 has N_c neurons without activation function. The wide shortcut, proposed in
 296 [46], enables neural networks to directly learn simple rules via it besides deep
 297 patterns through hidden layers, which is motivated by the fact that the inputs,
 298 such as P and T , are directly involved in the calculation of K_i . The concat
 299 layer concatenates the input layer and the outputs of the last residual block (the
 300 concatenation means putting two matrices $A \in \mathbb{R}^{d_1 \times d_2}$ and $B \in \mathbb{R}^{d_1 \times d_3}$ together
 301 to form a new one $C \in \mathbb{R}^{d_1 \times (d_2 + d_3)}$). In addition, the targets of the initializer
 302 are $\ln K_i$ instead of K_i , since K_i varies in different orders of magnitude, which

303 hampers the training of the initializer, whereas $\ln K_i$ does not.

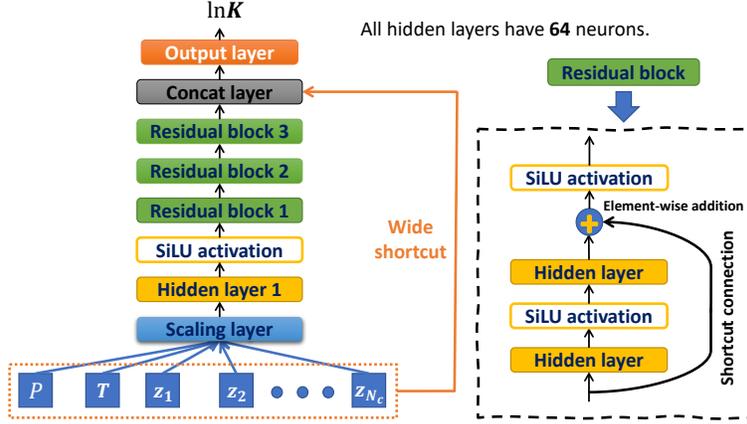


Figure 6: The architecture of initializer for the 9-component mixture

304 We found that the convergence of phase split calculations is robust if K_i
 305 predicted by the initializer can lead to more accurate values of the vapor fraction
 306 θ_V , especially around critical points where calculations are quite sensitive to
 307 initial K_i and prone to degenerate into trivial solutions. As a consequence, the
 308 loss function used to train the initializer consists of two parts, one is the mean
 309 absolute error (mae) in terms of K_i and the other is mae in terms of θ_V , as
 310 follows:

$$\text{mae}(\ln \mathbf{K}, \ln \hat{\mathbf{K}}) = \sum_{i=1}^{N_c} |\ln K_i - \ln \hat{K}_i| \quad (12a)$$

$$\text{mae}(\theta_V, \hat{\theta}_V) = |\theta_V - \hat{\theta}_V| \quad (12b)$$

311 where $\ln \mathbf{K}$ is the ground truth, $\ln \hat{\mathbf{K}}$ is the prediction of the initializer, θ_V is the
 312 vapor fraction at equilibrium, and $\hat{\theta}_V$ is obtained by solving the Rachford-Rice
 313 equation given \mathbf{z} and the prediction $\hat{\mathbf{K}}$.

314 4.2.2. Training

315 We generate one million samples in the two-phase region (K_i is not available
 316 at the monophasic region), which are divided into the training (70%), validation

317 (15%) and test (15%) sets. The training of the initializer is carried out in two
 318 stages. First, we train it to minimize $\text{mae}(\ln \mathbf{K}, \ln \hat{\mathbf{K}})$, using Adam with CLR
 319 and setting the batch size to 512. Second, after the above training, we further
 320 train it to minimize $\text{mae}(\ln \mathbf{K}, \ln \hat{\mathbf{K}}) + \text{mae}(\theta_V, \hat{\theta}_V)$, using Adam with a small
 321 learning rate $1.0e-5$. Here, $\partial \hat{\theta}_V / \partial \hat{\mathbf{K}}$ is required during backpropagation and can
 322 be simply computed via PyTorch’s automatic differentiation, which, however,
 323 differentiates through all the unrolled iterations, since we solve the Rachford-
 324 Rice equation in an iterative manner we described in Appendix C.1. Instead, we
 325 can make use of the implicit function theorem [47] to directly obtain $\partial \hat{\theta}_V / \partial \hat{\mathbf{K}}$
 326 by using the derivative information at the solution point of the Rachford-Rice
 327 equation, as follows:

$$\partial \hat{\theta}_V / \partial \hat{\mathbf{K}} = -[\partial_{\theta_V} f_{RR}(\hat{\theta}_V, \hat{\mathbf{K}})]^{-1} \partial_{\mathbf{K}} f_{RR}(\hat{\theta}_V, \hat{\mathbf{K}}) \quad (13)$$

328 This way is obviously more efficient and avoids differentiation through iterations.
 329 We give more details about the derivation of Equation 13 in Appendix C.2.

330 Eventually, the performance of the initializer on the test set is $\text{mae} = 9.66e-4$
 331 in terms of $\ln K_i$ and $\text{mae} = 1.86e-3$ in terms of K_i .

332 4.3. Strategy for accelerating flash calculation using neural networks

333 As shown in Figure 7, given P , T and \mathbf{z} , we first use the classifier to predict
 334 p . Next, based on two predefined thresholds, p_l and p_r , satisfying $p_l \leq p_r$,
 335 the given mixture is thought of as unstable if $p \leq p_l$ or stable if $p \geq p_r$. If
 336 $p_l < p < p_r$, we will use stability analysis to avoid unexpected errors. Here, we
 337 can adjust p_l and p_r to trade reliability for speed. In general, less errors occur
 338 with smaller p_l and greater p_r , but probably taking more time on stability
 339 analysis, and vice versa. A special case is $p_l = p_r = p_c$, where p_c could be a
 340 well-calibrated probability or simply set to 0.5, which means that we completely
 341 trust the classifier (i.e., stable if $p \geq p_c$ or unstable otherwise), and no extra
 342 stability analysis is required. For the initializer, it serves both stability analysis
 343 when $p_l < p < p_r$ and phase split calculations.

344 Neural networks can also be used individually. If only the classifier is avail-
 345 able, one may initialize K_i via the Wilson approximation rather than the ini-
 346 tializer in Figure 7. If only the initializer is available, one may use it to initialize
 347 K_i in Figure 2.

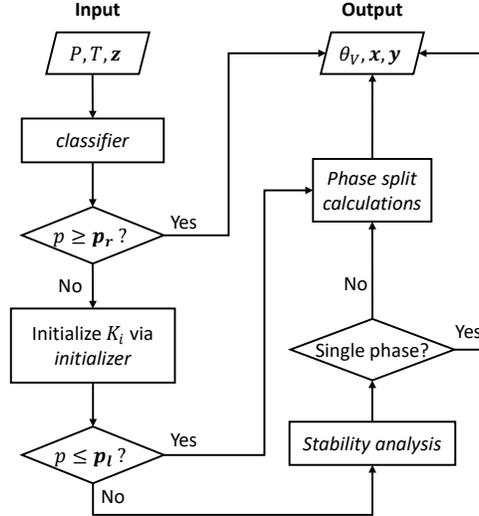


Figure 7: Acceleration of flash calculation using neural networks

348 5. Results

349 In this section, we will compare our proposed framework for vectorized flash
 350 calculation, *PTFlash*, with *Carnot*, an in-house thermodynamic library devel-
 351 oped by IFP Energies Nouvelles and based on C++. *Carnot* performs two-phase
 352 flash calculation in the manner shown in Figure 2, but can only handle samples
 353 one at a time on CPU. Regarding the hardware, CPU is Intel 11700F and GPU
 354 is NVIDIA RTX 3080 featuring 8704 CUDA cores and 10G memory. Note that
 355 since using multiple cores renders the frequency quite unstable due to heat ac-
 356 cumulation, we only use one core of CPU so that the frequency can be stabilized
 357 at 4.5GHz, which allows for a consistent criterion for measuring the execution
 358 time.

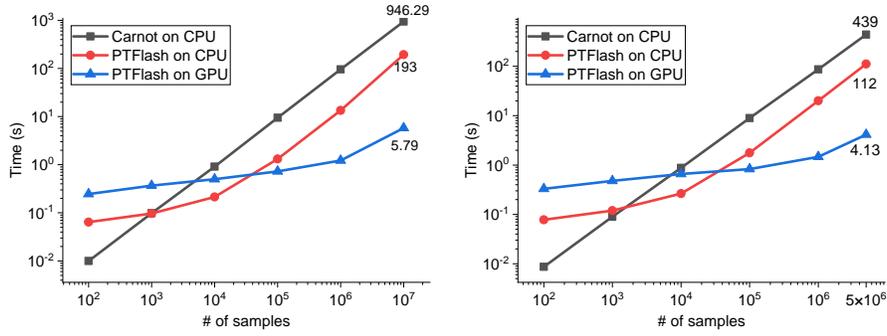
359 *PTFlash* and *Carnot* gave identical results (coincidence to 9 decimal places
360 under double-precision floating-point format) because they use exactly the same
361 convergence criteria for all iterative algorithms. In the following, we will focus
362 on comparing their speeds.

363 5.1. Vectorized flash calculation

364 We compare the execution time of different methods for flash calculation
365 with respect to the workload quantified by the number of samples n , as shown
366 in Figures 8. Due to GPU memory limitations, the maximum number of samples
367 allowed is 10, 5, and 1 million for the three case studies, respectively. We can
368 see that all three figures exhibit the same behavior. When the workload is
369 relatively low, e.g., $n < 1000$, *Carnot* wins by large margins, and CPU is also
370 preferable based on the fact that *PTFlash* runs much faster on CPU than on
371 GPU. On the one hand, PyTorch has some fixed overhead in the setup of the
372 working environment, e.g., the creation of tensors. On the other hand, when
373 GPU is used, there are some additional costs of CPU-GPU communication and
374 synchronization. When n is small, these overheads dominate. As proof, we can
375 see that the time of *PTFlash* on GPU hardly changes as n varies from 100 to
376 10^4 . In contrast, the time of *Carnot* is almost proportional to n .

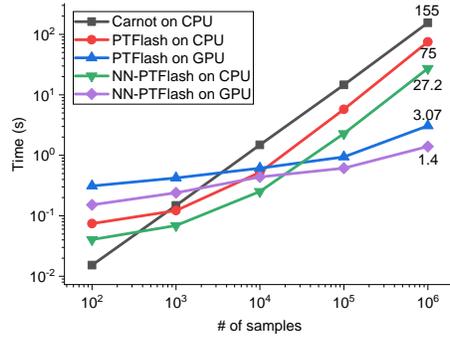
377 As the workload increases, the strength of *PTFlash* on GPU emerges and
378 becomes increasingly prominent. For the three case studies, *PTFlash* on GPU
379 is 163.4 (2 components), 106.3 (4 components) and 50.5 (9 components) times
380 faster than *Carnot* at the maximum number of samples. This suggests that
381 *PTFlash* on GPU is more suitable for large scale computation. Interestingly, we
382 can observe that *PTFlash* on CPU also outperforms *Carnot* when the workload
383 is relatively heavy, e.g., $n > 10^3$. In fact, thanks to Advanced Vector Extensions,
384 vectorization enables fuller utilization of CPU's computational power.

385 We notice that there is a lack of a comprehensive and unified benchmark for
386 the runtime of flash calculation in the literature. Here, we give an article with
387 a case study similar to ours for readers' reference [48], which claimed that the
388 total computation time of flash calculations is 10 seconds for one million samples



(a) Mixture of CH_4 and C_6H_{14}

(b) Mixture of 4 components



(c) Mixture of 9 components

Figure 8: Comparison between *PTFlash* and *Carnot* in terms of speed. *NN-PTFlash* is *PTFlash* accelerated by neural networks, as presented in Section 4.

389 of a 9-component mixture. However, it is worth pointing out that the sampling
 390 method, convergence criteria and algorithm implementation in this reference
 391 article are different from ours. In our work, these aspects are consistent for
 392 both *Carnot* and *PTFlash* to ensure a fair comparison.

393 Next, we focus on the mixture of 9 components and analyze the performance
 394 of *PTFlash* for this case study. Table 4 is a performance profiler of *PTFlash*
 395 on GPU at $n = 10^6$, which records the running time of each subroutine of
 396 flash calculations. As a complement, Figures 9 dissect phase split calculations
 397 by tracking the total elapsed time and the convergence percentage up to each
 398 iteration, as well as the mean of critical distances d_c of converged samples at

399 each iteration, where d_c is defined as:

$$d_c = \sqrt{\sum_{i=1}^{N_c} \ln K_i^2} \quad (14)$$

400 The closer to critical points, the smaller d_c . In other words, d_c indicates the
 401 closeness to critical points.

402 The observations of Figures 9 are summarized as follows: (1) In Figure 9(a),
 403 the slope of time with respect to the number of iterations is decreasing because
 404 the workload is reduced due to incremental convergence. (2) In Figure 9(b), for
 405 the samples that do not converge after successive substitution, the majority of
 406 them (92.67%) converge after 3 iterations of the trust-region method. (3) In
 407 Figure 9(c), d_c decreases during iterations, which means that samples close to
 408 critical points converge last and also confirms that convergence is slow around
 409 critical points.

Table 4: Performance profiler of *PTFlash* on GPU (Figure 2) for the mixture of 9 components at $n = 10^6$ in Figure 8(c)

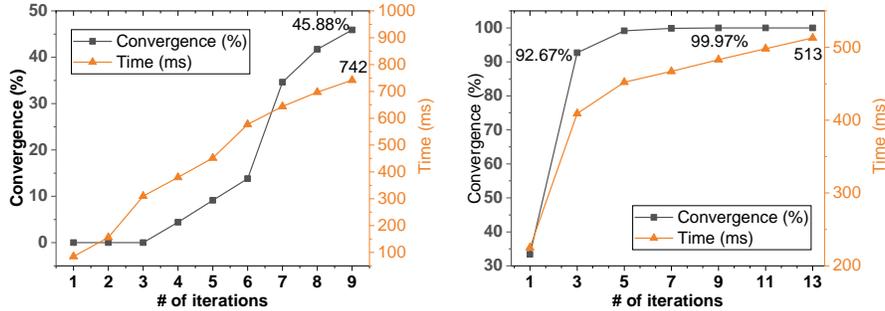
	ss of phase split calculations	Stability analysis				Phase split	
		vapor-like estimate		liquid-like estimate		calculations	
		ss	tr	ss	tr	ss	tr
# of samples	10^6	625645	130715	625645	90179	413442	223741
Convergence	37.44% ¹	79.11%	100%	85.59%	100%	45.88%	100%
Max number of iterations	3	9	18	9	16	9	13
Total time	0.4565s	0.4136s	0.3417s	0.4044s	0.2706s	0.7412s	0.5132s
		1.3237s ²				1.2544s	

ss: successive substitution tr: trust-region method

¹ 37.44% is the percentage of samples for which any of ΔG , tpd_x and tpd_y is negative after 3 attempts of successive substitution, as described in Section 2.2.3.

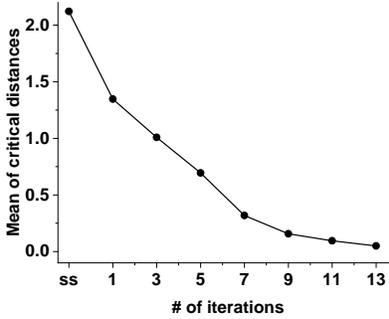
² The total time of stability analysis is less than the sum of the times of all subroutines because vapor-like and liquid-like estimates are handled concurrently.

410 The above analysis gives us a general understanding of *PTFlash*, but in fact



(a) Successive substitution

(b) Trust-region method



(c) Closeness to critical points

Figure 9: Figures (a) and (b) show the convergence percentage and the elapsed time up to each iteration of phase split calculations of *PTFlash* on GPU. In Figure (c), on the x-axis, ss corresponds to the end of successive substitution and other integers are the number of iterations of the trust-region method.

411 it is not easy to analyze *PTFlash* comprehensively because each subroutine also
 412 contains iterative algorithms, such as solving the SRK equation of state and the
 413 Rachford-Rice equation. Nevertheless, given the information already obtained,
 414 we know that we need to shorten the time of stability analysis and reduce the
 415 number of iterations in order to accelerate *PTFlash*, which is exactly the role
 416 of the classifier and initializer.

417 5.2. Deep-learning-powered vectorized flash calculation

418 We trained neural networks following Section 4 for the mixture of 9 compo-
 419 nents. Here, we will explore the effect of neural networks. First of all, we set

420 $p_l = 0.02$ and $p_r = 0.98$ as the thresholds of stability and instability, which are
 421 carefully chosen so that no misclassification occurs. In Figure 8(c), we can see
 422 that *NN-PTFlash* outpaces *PTFlash* on both CPU (2.7x speed-up) and GPU
 423 (2.2x speed-up). In addition, *NN-PTFlash* on GPU runs almost 110.7 times
 424 faster than *Carnot* at $n = 10^6$.

425 Table 5 is the performance profiler of *NN-PTFlash* on GPU. We can see that
 426 the classifier is able to precisely determine the stability of the vast majority of
 427 samples (99.42%), which significantly relieves the burden of stability analysis
 428 and saves time. In addition, compared to phase split calculations of *PTFlash*,
 429 the convergence percentage of successive substitution increases from 45.88% to
 430 67.40%, and the overall time is also greatly reduced, which is attributed to
 431 better initial K_i provided by the initializer.

Table 5: Performance profiler of *NN-PTFlash* on GPU (Figure 7) for the mixture of 9 components at $n = 10^6$ in Figure 8(c)

	classifier	Stability analysis				Phase split	
		vapor-like estimate		liquid-like estimate		calculations	
		ss	tr	ss	tr	ss	tr
# of samples	10^6	5818	1073	5818	1704	413442	134786
Convergence	99.42% ¹	81.56%	100%	70.71%	100%	67.40%	100%
Max number of iterations		9	13	9	12	9	13
Total time	0.0005s	0.1365s	0.128s	0.0514s	0.12s	0.7043s	0.3388s
		0.34s				1.0431s	

ss: successive substitution tr: trust-region method

¹ 99.42% includes 58.38% predicted as stable (i.e., $p > p_r$) and 41.04% predicted as unstable (i.e., $p < p_l$).

432 We also performed ablation studies to compare the contributions of the clas-
 433 sifier and initializer by using them individually. For instance, when handling 1
 434 million samples for the case study containing 9 components, *NN-PTFlash* with
 435 only the classifier on GPU takes 1.88s. However, the attempt to use the ini-

436 tializer alone fails because we found its outputs may reach unreasonably large
437 values (e.g., $1.0e15$) for stable mixtures far away from the boundary between the
438 single-phase and two-phase regions, which leads to numerical overflow. From
439 machine learning terminology, this is the out-of-distribution generalization prob-
440 lem, since the initializer is trained on the two-phase region and may suffer from
441 large predictive errors when used within the single-phase region. Nonetheless,
442 there is no problem when the initializer works in tandem with the classifier be-
443 cause remaining samples located in the single-phase region are fairly close to the
444 boundary after filtering through the classifier, as shown in Figure 5(b). In any
445 case, based on the fact that *NN-PTFlash* using only the classifier always lags
446 behind that using both, we can conclude that both the classifier and initializer
447 play an important role in speeding up flash calculations.

448 5.3. Discussion

449 The results show that the systematic and exhaustive vectorization of two-
450 phase flash calculation does result in attractive speed-ups when large scale com-
451 putation is involved, e.g., the number of samples to process is on the order of
452 millions. Importantly, this speed-up does not come at the cost of accuracy and
453 stability like [11, 12, 14, 15] which are subject to the unreliability of machine
454 learning models. In addition, we can see that neural networks, such as the
455 classifier and initializer, really make a big difference.

456 Due to GPU memory limitations, the number of samples n is limited in
457 Figures 8. Nonetheless, we can see that the slopes of time with respect to n differ
458 significantly between different methods. The time of *Carnot* is proportional to
459 n , in contrast, the time of *PTFlash* on GPU is increasing slowly. Therefore,
460 it is reasonable to believe that the speed advantage of *PTFlash* on GPU will
461 become increasingly prominent if n continues to grow.

462 Using PyTorch has several benefits in addition to its simplicity and flexibil-
463 ity. First, we can seamlessly incorporate neural networks into *PTFlash*. Second,
464 any subroutine of *PTFlash* is fully differentiable through automatic differenti-
465 ation, and we can also leverage the implicit function theorem for efficient dif-

466 ferentiation, as we did in Section 4.2.2. Third, PyTorch’s highly optimized
467 and ready-to-use multi-GPU parallelization largely circumvents the painstaking
468 hand-crafted effort.

469 *PTFlash* also has several limitations. First, *PTFlash* is based on the SRK
470 equation of state, which is relatively simple and sufficient for mixtures contain-
471 ing hydrocarbons and non-polar components, but does not take into account
472 the effect of hydrogen bonding and falls short of adequacy for cross-associating
473 mixtures having polar components, such as water and alcohol [49]. In this
474 case, more advanced but also more complicated equations of state should be
475 employed, such as the SAFT equation of state [50–55] or the CPA equation of
476 state [56, 57]. However, vectorization of these complicated equations of state is
477 far more difficult than that of cubic equations of state. To alleviate this prob-
478 lem, we plan to use neural networks to directly predict the fugacity coefficients.
479 In this way, we can calculate the fugacity coefficients in a vectorized fashion,
480 regardless of the equation of state used. Second, *PTFlash* consumes a large
481 amount of GPU memory, badly limiting its use on much larger batches of data.
482 We need to optimize *PTFlash* to reduce the consumption of GPU memory, e.g.,
483 by leveraging the sparsity and symmetry of matrices. Third, *PTFlash* does
484 not support multi-phase equilibrium. Last but not least, neural networks are
485 subject to the out-of-distribution generalization problem. If pressure and tem-
486 perature are out of predefined ranges used to train neural networks, predictive
487 performance will deteriorate dramatically. Furthermore, once the components
488 of the mixture change, we need to create new neural networks and train them
489 from scratch.

490 6. Conclusion

491 In this work, we presented a fast and parallel framework, *PTFlash*, for two-
492 phase flash calculation based on PyTorch, which efficiently vectorizes algorithms
493 and gains attractive speed-ups at large scale calculations. Two neural networks
494 were used to predict the stability of given mixtures and to initialize the distribu-

495 tion coefficients more accurately than the Wilson approximation, which greatly
496 accelerate *PTFlash*. In addition, *PTFlash* has much broader utility compared to
497 the aforementioned methods which are mainly tailored to compositional reser-
498 voir simulation.

499 We compared *PTFlash* with *Carnot*, an in-house thermodynamic library,
500 and we investigated three case studies containing 2, 4 and 9 components with
501 maximum number of samples of 10, 5 and 1 million, respectively. The results
502 showed that *PTFlash* on GPU is 163.4, 106.3 and 50.5 times faster than *Carnot*
503 at the maximum number of samples for these three cases, respectively.

504 In the future, we will optimize *PTFlash* to reduce the consumption of GPU
505 memory and extend our work to vectorize more advanced equations of state and
506 support multi-phase equilibrium. We will also explore the feasibility of using
507 neural networks to directly predict the fugacity coefficients, which can serve as
508 an alternative to numerically solving equations of state. In addition, we will
509 validate *PTFlash* on more hardware suitable for parallel computing, e.g., TPU.
510 Last but not least, we will apply our work to downstream applications, e.g.,
511 compositional reservoir simulation.

512 **7. Acknowledgements**

513 We acknowledge the financial support from French National Research Agency
514 (ANR) through the projects DL4CLIM ANR-19-CHIA-0018-01 and DEEP-
515 NUM ANR-21-CE23-0017-02.

516 **Appendix A. SRK equation of state and its solution**

517 The SRK equation of state describes the relationship between pressure (P),
 518 temperature (T) and volume (V) in the following mathematical form [21]:

$$P = \frac{RT}{V - b} - \frac{a\alpha}{V(V + b)} \quad (\text{A.1})$$

519 where R is the gas constant, $a\alpha$ refers to the temperature-dependent energy
 520 parameter, and b denotes the co-volume parameter. We employ the van der
 521 Waals mixing rules and the classical combining rules to calculate $a\alpha$ and b , as
 522 follows:

$$a\alpha = \sum_{i=1}^{N_c} \sum_{j=1}^{N_c} c_i c_j (a\alpha)_{ij} \quad (\text{A.2a})$$

$$(a\alpha)_{ij} = (1 - k_{ij}) \sqrt{(a\alpha)_i (a\alpha)_j} \quad (\text{A.2b})$$

$$b = \sum_{i=1}^{N_c} c_i b_i \quad (\text{A.2c})$$

$$a_i = \frac{0.42748 \cdot R^2 (T_{c,i})^2}{P_{c,i}} \quad (\text{A.2d})$$

$$b_i = \frac{0.08664 \cdot R T_{c,i}}{P_{c,i}} \quad (\text{A.2e})$$

$$\alpha_i = \left[1 + m_i \left(1 - \sqrt{\frac{T}{T_{c,i}}} \right) \right]^2 \quad (\text{A.2f})$$

$$m_i = 0.480 + 1.574 \omega_i - 0.176 \omega_i^2 \quad (\text{A.2g})$$

523 where the subscripts i and j refer to the components i and j , respectively, c_i
 524 denotes the mole fraction of the component i in the phase considered, k_{ij} is the
 525 binary interaction parameter between the components i and j , a_i and b_i are two
 526 substance-specific constants related to the critical temperature $T_{c,i}$ and critical
 527 pressure $P_{c,i}$, and ω_i is the acentric factor. We reformulate Equation A.1 as a
 528 cubic equation in terms of the compressibility factor Z :

$$f_{srk}(Z) = Z^3 - Z^2 + \rho_1 Z - \rho_0 = 0 \quad (\text{A.3})$$

529 where $\rho_0 = AB$ and $\rho_1 = A - B(1 + B)$, in which $A = a\alpha P/(R^2T^2)$ and
 530 $B = bP/(RT)$. To find the roots of $f_{srk}(Z)$, we utilize an iterative approach
 531 based on Halley’s method [25], as follows:

$$Z^{(k+1)} = Z^{(k)} - \frac{f_{srk}(Z^{(k)})}{f'_{srk}(Z^{(k)})} \left[1 - \frac{f_{srk}(Z^{(k)})}{f'_{srk}(Z^{(k)})} \cdot \frac{f''_{srk}(Z^{(k)})}{2f'_{srk}(Z^{(k)})} \right]^{-1} \quad (\text{A.4})$$

532 The above iteration starts with a liquid-like guess and converges to a real
 533 root Z_0 (The convergence criterion is $|Z^{(k+1)}/Z^{(k)} - 1| < 1.0e-8$), and then we
 534 deflate the cubic equation as:

$$f_{srk}(Z) = (Z - Z_0)(Z^2 + pZ + q) = 0 \quad (\text{A.5})$$

535 where $p = Z_0 - 1$ and $q = pZ_0 + \rho_1$. If $p^2 < 4q$, only one real root Z_0 exists,
 536 otherwise, there are three real roots and the other two are $-p/2 \pm \sqrt{p^2 - 4q}/2$. In
 537 the latter case, we assign the smallest root to the liquid phase and the biggest
 538 one to the vapor phase. Subsequently, the root corresponding to the lowest
 539 Gibbs energy will be chosen. When Z is known, the fugacity coefficients φ_i are
 540 calculated as follows:

$$\begin{aligned} \ln \varphi_i(P, T, \mathbf{c}) &= \frac{b_i}{b} (Z - 1) - \ln(Z - B) \\ &+ \frac{A}{B} \left(\frac{b_i}{b} - \frac{2}{a\alpha} \sum_{j=1}^{N_c} (a\alpha)_{ij} c_j \right) \ln\left(1 + \frac{B}{Z}\right) \end{aligned} \quad (\text{A.6})$$

541 where \mathbf{c} is the composition of the phase considered. In addition, the derivatives
 542 of the fugacity coefficients with respect to mole numbers, which are necessary for
 543 the trust-region methods of stability analysis and phase split calculations, are
 544 calculated explicitly rather than through PyTorch’s automatic differentiation,
 545 which requires retaining intermediate results and consumes prohibitive memory
 546 at large scale computation.

547 **Appendix B. Trust-region method**

548 When the successive substitution fails to converge quickly, particularly around
 549 critical points for which liquid and vapor phases are almost indistinguishable, we
 550 will switch to the trust-region method with restricted steps, which is a second-
 551 order optimization technique, to achieve faster convergence.

552 In the following, the problem formulations are taken from Michelsen and
 553 Mollerup's book [31], but the concrete implementation of the trust-region method,
 554 such as how to adjust the trust-region size and calculate the step size, is adapted
 555 from [28].

556 *Appendix B.1. Trust-region method for stability analysis*

557 The objective function to be minimized is the modified tangent plane dis-
 558 tance [1]:

$$tm(\mathbf{W}) = \sum_{i=1}^{N_c} W_i (\ln W_i + \ln \varphi_i(\mathbf{W}) - \ln z_i - \ln \varphi_i(\mathbf{z}) - 1)$$

559 The minimization is accomplished by iterating the following equations:

$$\boldsymbol{\beta}^{(k)} = 2\sqrt{\mathbf{W}^{(k)}} \quad (\text{B.1a})$$

$$(\mathbf{H}^{(k)} + \eta^{(k)}\mathbf{I}) \cdot \Delta\boldsymbol{\beta} + \mathbf{g}^{(k)} = \mathbf{0} \quad \text{s.t.} \quad \|\Delta\boldsymbol{\beta}\| \leq \Delta_{max}^{(k)} \quad (\text{B.1b})$$

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \Delta\boldsymbol{\beta} \quad (\text{B.1c})$$

$$\mathbf{W}^{(k+1)} = \left(\frac{\boldsymbol{\beta}^{(k+1)}}{2} \right)^2 \quad (\text{B.1d})$$

560 where \mathbf{I} is the identity matrix, \mathbf{g} and \mathbf{H} are the gradient and Hessian matrix
 561 of tm with respect to $\boldsymbol{\beta}$, respectively, and are calculated as follows:

$$g_i = \sqrt{W_i} (\ln W_i + \ln \varphi_i(\mathbf{W}) - \ln z_i - \ln \varphi_i(\mathbf{z})) \quad (\text{B.2a})$$

$$H_{ij} = \sqrt{W_i W_j} \frac{\partial \ln \varphi_i}{\partial W_i} + \sigma_{ij} \left(1 + \frac{g_i}{\beta_i} \right) \quad \text{where } \sigma_{ij} = 1 \Leftrightarrow i = j \quad (\text{B.2b})$$

562 In addition, η is the trust-region size used to guarantee the positive definite-
563 ness of $\mathbf{H} + \eta\mathbf{I}$ and to tailor the step size to meet $\|\Delta\boldsymbol{\beta}\| \leq \Delta_{max}$, where Δ_{max}
564 is adjusted during iterations depending on the match between the actual reduc-
565 tion $\delta_{tm} = tm^{(k+1)} - tm^{(k)}$ and the predicted reduction based on the quadratic
566 approximation $\hat{\delta}_{tm} = \Delta\boldsymbol{\beta}^T \mathbf{g} + \frac{1}{2} \Delta\boldsymbol{\beta}^T \mathbf{H} \Delta\boldsymbol{\beta}$, using the following heuristic rules:

$$\Delta_{max}^{(k+1)} = \begin{cases} \frac{\Delta_{max}^{(k)}}{2}, & \text{if } \left| \delta_{tm} / \hat{\delta}_{tm} \right| \leq 0.25 \\ 2\Delta_{max}^{(k)}, & \text{if } \left| \delta_{tm} / \hat{\delta}_{tm} \right| \geq 0.75 \\ \Delta_{max}^{(k)}, & \text{otherwise} \end{cases} \quad (\text{B.3})$$

567 The convergence criterion of Equation B.1 is $\max(|\mathbf{g}|) < 1.0e-6$.

568 *Appendix B.2. Trust-region method for phase split calculations*

569 The objective function to be minimized is the reduced Gibbs energy:

$$G = \sum_{i=1}^{N_c} n_i^L (\ln x_i + \ln \varphi_i^L) + \sum_{i=1}^{N_c} n_i^V (\ln y_i + \ln \varphi_i^V)$$

570 where $n_i^L = x_i(1 - \theta_V)$ and $n_i^V = y_i\theta_V$ are the mole numbers of liquid and vapor
571 phases, respectively. We choose n_i^V as the independent variable and perform
572 the following iteration:

$$\left(\tilde{\mathbf{H}}^{(k)} + \tilde{\boldsymbol{\eta}}^{(k)} \cdot \mathbf{D} \left(\frac{\mathbf{z}}{\mathbf{xy}} \right) \right) \cdot \Delta \mathbf{n}^V + \tilde{\mathbf{g}}^{(k)} = \mathbf{0} \quad \text{s.t.} \quad \|\Delta \mathbf{n}^V\| \leq \tilde{\Delta}_{max}^{(k)} \quad (\text{B.4a})$$

$$\mathbf{n}^{V,k+1} = \mathbf{n}^{V,k} + \Delta \mathbf{n}^V \quad (\text{B.4b})$$

573 where $\tilde{\mathbf{H}}^{(k)}$ and $\tilde{\mathbf{g}}^{(k)}$ are the gradient and hessian matrix of G with respect to
574 n_i^V , respectively, and are calculated as follows:

$$\tilde{g}_i = \ln y_i + \ln \varphi_i^V - \ln x_i - \ln \varphi_i^L \quad (\text{B.5a})$$

$$\tilde{H}_{ij} = \frac{1}{\theta_V(1 - \theta_V)} \left(\frac{z_i}{x_i y_i} \sigma_{ij} - 1 + \theta_V \frac{\partial \ln \varphi_i^L}{\partial n_j^L} + (1 - \theta_V) \frac{\partial \ln \varphi_i^V}{\partial n_j^V} \right) \quad (\text{B.5b})$$

575 In addition, $\mathbf{D}(\cdot)$ is a diagonal matrix with diagonal entries in parentheses.
576 The above iteration stops if $\max(|\tilde{\mathbf{g}}|) < 1.0e-8$. Here, the trust-region method is
577 implemented in the same way as in stability analysis.

578 **Appendix C. The Rachford-Rice equation**

579 *Appendix C.1. Solution of the Rachford-Rice equation*

580 The Rachford-Rice equation is as follows:

$$f_{RR}(\theta_V, \mathbf{K}) = \sum_{i=1}^{N_c} \frac{(K_i - 1)z_i}{1 + (K_i - 1)\theta_V} = 0$$

581 Given \mathbf{K} , the solution of the above equation amounts to finding an appropriate
 582 zero yielding all non-negative phase compositions. Concretely, we adopt the
 583 method proposed by [30], which transforms f_{RR} into a helper function h_{RR}
 584 which is more linear in the vicinity of the zero:

$$h_{RR}(\theta_V, \mathbf{K}) = (\theta_V - \alpha_l) \cdot (\alpha_r - \theta_V) \cdot f_{RR}(\theta_V) = 0 \quad (\text{C.1})$$

585 where $\alpha_l = 1/(1 - \max(K_i))$ and $\alpha_r = 1/(1 - \min(K_i))$. The above equation
 586 is solved by alternating between the Newton method and the bisection method
 587 used when the Newton step renders θ_V out of the bounds which contain the zero
 588 and become narrower during iterations. When the Newton step size is smaller
 589 than $1.0e-8$, the iteration stops.

590 *Appendix C.2. Calculation of $\partial\theta_V/\partial\mathbf{K}$ using the implicit function theorem*

591 Based on the implicit function theorem [47], we can calculate $\partial\theta_V/\partial\mathbf{K}$ in
 592 an efficient way. We first differentiate the Rachford-Rice equation with respect
 593 to \mathbf{K} (note that θ_V is an implicit function of \mathbf{K}) and get:

$$\partial_{\theta_V} f_{RR}(\theta_V, \mathbf{K}) \times \partial\theta_V/\partial\mathbf{K} + \partial_{\mathbf{K}} f_{RR}(\theta_V, \mathbf{K}) = 0 \quad (\text{C.2})$$

594 We rearrange the above equation and get Equation 13, as follows:

$$\partial\theta_V/\partial\mathbf{K} = -[\partial_{\theta_V} f_{RR}(\theta_V, \mathbf{K})]^{-1} \partial_{\mathbf{K}} f_{RR}(\theta_V, \mathbf{K})$$

595 Moreover, since $\partial_{\theta_V} f_{RR}(\theta_V, \mathbf{K})$ is a scalar, we can further reduce the above
 596 equation to:

$$\partial\theta_V/\partial\mathbf{K} = -\frac{\partial_{\mathbf{K}} f_{RR}(\theta_V, \mathbf{K})}{\partial_{\theta_V} f_{RR}(\theta_V, \mathbf{K})} \quad (\text{C.3})$$

597 For the sake of brevity, we have simplified some details. For more details and a
 598 defense of the above derivation, refer to [47].

599 **Appendix D. Some typical reservoir fluid compositions**

Table D.6: Some typical reservoir fluid compositions

	Wet gas	Gas condensate	Volatile oil	Black oil
CH_4	92.46%	73.19%	57.6%	33.6%
C_2H_6	3.18%	7.8%	7.35%	4.01%
C_3H_8	1.01%	3.55%	4.21%	1.01%
$n-C_4H_{10}$	0.52%	2.16%	2.81%	1.15%
$n-C_5H_{12}$	0.21%	1.32%	1.48%	0.65%
C_6H_{14}	0.14%	1.09%	1.92%	1.8%
$C_7H_{16}^+$	0.82%	8.21%	22.57%	57.4%
CO_2	1.41%	2.37%	1.82%	0.07%
N_2	0.25%	0.31%	0.24%	0.31%

600 **Appendix E. Vectorized algorithms**

601 *Appendix E.1. Synchronizer*

Algorithm 1: PyTorch pseudo-code of *synchronizer* to save converged results after iteration and remove the corresponding samples

Input: Vectorized iterated function $f(\mathbf{X}, \mathbb{O})$, initial estimate $\mathbf{X}^{(0)}$, other f -related inputs \mathbb{O} , convergence criterion C , maximum number of iterations K

1 **Initialization**

2 Set the number of iterations $k \leftarrow 1$

3 Generate a vector i containing indices from 0 to $n - 1$

 /* n is the number of samples and indexing starts from 0. */

4 Create a placeholder matrix $\widetilde{\mathbf{X}}$ of the same shape as $\mathbf{X}^{(0)}$

5 **while** $k \leq K$ **do**

6 $\mathbf{X}^{(k+1)} \leftarrow f(\mathbf{X}^{(k)}, \mathbb{O})$

7 $\text{mask} \leftarrow C(\dots)$

 /* C returns a Boolean vector and True means convergence. */

8 **Saving**

9 indices $\leftarrow i[\text{mask}]$

10 $\widetilde{\mathbf{X}}[\text{indices}] \leftarrow \mathbf{X}^{(k+1)}[\text{mask}]$

11 **Removing**

12 $i \leftarrow i[\sim \text{mask}]$

13 $\mathbb{O} \leftarrow \mathbb{O}[\sim \text{mask}]$

 /* Apply this operation to every element in \mathbb{O} */

14 $\mathbf{X}^{(k+1)} \leftarrow \mathbf{X}^{(k+1)}[\sim \text{mask}]$

15 $k \leftarrow k + 1$

16 **if** $\text{len}(i) \neq 0$ **then**

17 $\widetilde{\mathbf{X}}[i] \leftarrow \mathbf{X}$

 /* Also save unconverged results for further utilization. */

Output: Converged results $\widetilde{\mathbf{X}}$ and unconverged indices i

Algorithm 2: PyTorch pseudo-code of vectorized stability analysis

Input: Pressure \mathbf{P} , temperature \mathbf{T} , feed composition \mathbf{z} , component properties ($\mathbf{P}_c, \mathbf{T}_c, \boldsymbol{\omega}$, BIPs), initial estimate $\mathbf{W}^{(0)}$, convergence criteria C_{ss} and C_{tr} , maximum numbers of iterations $K_{ss} = 9$ and $K_{tr} = 20$

- 1 **Initialization**
- 2 Instantiate $pteos = PTEOS(\mathbf{P}_c, \mathbf{T}_c, \boldsymbol{\omega}, \text{BIPs})$
 /* *PTEOS* is a PyTorch-based class to efficiently calculate the
 fugacity coefficients and their partial derivatives. */
- 3 **Successive substitution**
- 4 Iterated function f_{ss} specified by Equation 5
- 5 Other inputs $\mathbb{O}_{ss} \leftarrow \{\mathbf{P}, \mathbf{T}, \mathbf{z}\}$
- 6 $\mathbf{W}, \mathbf{i}_{ss} \leftarrow \text{synchronizer}(f_{ss}, \mathbf{W}^{(0)}, \mathbb{O}_{ss}, \mathbb{C}_{ss}, K_{ss})$
- 7 **Trust-region method**
- 8 Iterated function f_{tr} specified by Equation B.1
- 9 $\mathbf{W}_{tr}^{(0)} \leftarrow \mathbf{W}[\mathbf{i}_{ss}]$
- 10 Other inputs $\mathbb{O}_{tr} \leftarrow \{\mathbf{P}[\mathbf{i}_{ss}], \mathbf{T}[\mathbf{i}_{ss}], \mathbf{z}[\mathbf{i}_{ss}]\}$
- 11 $\mathbf{W}_{tr}, \mathbf{i}_{tr} \leftarrow \text{synchronizer}(f_{tr}, \mathbf{W}_{tr}^{(0)}, \mathbb{O}_{tr}, \mathbb{C}_{tr}, K_{tr})$
- 12 $\mathbf{W}[\mathbf{i}_{ss}] \leftarrow \mathbf{W}_{tr}$ and $\mathbf{i} \leftarrow \mathbf{i}_{ss}[\mathbf{i}_{tr}]$

Output: Converged results \mathbf{W} and unconverged indices \mathbf{i}

603 **References**

- 604 [1] M. L. Michelsen, The isothermal flash problem. part II. phase-split calcu-
605 lation 9 (1) 21–40, publisher: Elsevier.
- 606 [2] M. L. Michelsen, The isothermal flash problem. part i. stability 9 (1) 1–19,
607 publisher: Elsevier.
- 608 [3] P. Wang, E. H. Stenby, Non-iterative flash calculation algorithm in com-
609 positional reservoir simulation 95 93–108, publisher: Elsevier.
- 610 [4] A. Belkadi, W. Yan, M. L. Michelsen, E. H. Stenby, Comparison of two
611 methods for speeding up flash calculations in compositional simulations,
612 in: SPE Reservoir Simulation Symposium, OnePetro.
- 613 [5] A. H. Dogru, L. S. K. Fung, U. Middy, T. Al-Shaalan, J. A. Pita, A
614 next-generation parallel reservoir simulator for giant reservoirs, in: SPE
615 Reservoir Simulation Symposium, OnePetro.
- 616 [6] M. L. Michelsen, Simplified flash calculations for cubic equations of state
617 25 (1) 184–188, publisher: ACS Publications.
- 618 [7] E. M. Hendriks, Reduction theorem for phase equilibrium problems 27 (9)
619 1728–1732, publisher: ACS Publications.
- 620 [8] E. M. Hendriks, A. Van Bergen, Application of a reduction method to phase
621 equilibria calculations 74 17–34, publisher: Elsevier.
- 622 [9] C. P. Rasmussen, K. Krejbjerg, M. L. Michelsen, K. E. Bjurstrøm, In-
623 creasing the computational speed of flash calculations with applications for
624 compositional, transient simulations 9 (1) 32–38, publisher: OnePetro.
- 625 [10] D. Voskov, H. A. Tchelepi, Compositional space parameterization for flow
626 simulation, in: SPE Reservoir Simulation Symposium, OnePetro.
- 627 [11] V. Gaganis, N. Varotsis, Machine learning methods to speed up compo-
628 sitional reservoir simulation, in: SPE Europec/EAGE annual conference,
629 OnePetro.

- 630 [12] V. Gaganis, N. Varotsis, An integrated approach for rapid phase behavior
631 calculations in compositional modeling 118 74–87, publisher: Elsevier.
- 632 [13] V. Gaganis, Rapid phase stability calculations in fluid flow simulation using
633 simple discriminating functions, *Computers & Chemical Engineering* 108
634 (2018) 112–127.
- 635 [14] A. Kashinath, M. Szulczewski, A. Dogru, A fast algorithm for calculating
636 isothermal phase behavior using machine learning 465 73–82. doi:10.
637 1016/j.fluid.2018.02.004.
- 638 [15] S. Wang, N. Sobecki, D. Ding, L. Zhu, Y.-S. Wu, Accelerating and sta-
639 bilizing the vapor-liquid equilibrium (VLE) calculation in compositional
640 simulation of unconventional reservoirs using deep learning based flash cal-
641 culation 253 209–219. doi:10.1016/j.fuel.2019.05.023.
- 642 [16] T. Zhang, Y. Li, Y. Li, S. Sun, X. Gao, A self-adaptive deep learning
643 algorithm for accelerating multi-component flash calculation, *Computer
644 Methods in Applied Mechanics and Engineering* 369 (2020) 113207.
- 645 [17] C. Cortes, V. Vapnik, Support-vector networks 20 (3) 273–297, publisher:
646 Springer.
- 647 [18] M. E. Tipping, Sparse bayesian learning and the relevance vector machine
648 1 211–244.
- 649 [19] I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*, MIT press.
- 650 [20] Z. Chen, H. Liu, S. Yu, B. Hsieh, L. Shao, GPU-based parallel reservoir
651 simulators, in: *Domain Decomposition Methods in Science and Engineering
652 XXI*, Springer, pp. 199–206.
- 653 [21] G. Soave, Equilibrium constants from a modified redlich-kwong equation
654 of state 27 (6) 1197–1203, publisher: Elsevier.

- 655 [22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan,
656 T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, others, Pytorch: An im-
657 perative style, high-performance deep learning library 32 8026–8037.
- 658 [23] C. Lomont, Introduction to intel advanced vector extensions 23.
- 659 [24] J. Sanders, E. Kandrot, CUDA by example: an introduction to general-
660 purpose GPU programming, Addison-Wesley Professional.
- 661 [25] U. K. Deiters, R. Macías-Salinas, Calculation of densities from cubic equa-
662 tions of state: revisited, Industrial & Engineering Chemistry Research
663 53 (6) (2014) 2529–2536.
- 664 [26] Y. Zhi, H. Lee, Fallibility of analytic roots of cubic equations of state in
665 low temperature region 201 (2) 287–294, publisher: Elsevier.
- 666 [27] O. Orbach, C. Crowe, Convergence promotion in the simulation of chemical
667 processes with recycle-the dominant eigenvalue method 49 (4) 509–513,
668 publisher: Wiley Online Library.
- 669 [28] M. Hebden, An algorithm for minimization using exact second deriva-
670 tivesPublisher: Citeseer.
- 671 [29] H. H. Rachford, J. Rice, Procedure for use of electronic digital comput-
672 ers in calculating flash vaporization hydrocarbon equilibrium, Journal of
673 Petroleum Technology 4 (10) (1952) 19–3.
- 674 [30] C. Leibovici, J. Neoschil, A new look at the rachford-rice equation 74 303–
675 308. doi:10.1016/0378-3812(92)85069-K.
- 676 [31] M. L. Michelsen, J. Mollerup, Thermodynamic modelling: fundamentals
677 and computational aspects, Tie-Line Publications.
- 678 [32] M. D. McKay, R. J. Beckman, W. J. Conover, A comparison of three
679 methods for selecting values of input variables in the analysis of output
680 from a computer code 42 (1) 55–61, publisher: Taylor & Francis.

- 681 [33] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin,
682 G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang,
683 JAX: composable transformations of python+NumPy programs.
684 URL <http://github.com/google/jax>
- 685 [34] G. Van Rossum, F. L. Drake, The python language reference manual, Net-
686 work Theory Ltd.
- 687 [35] D. Hendrycks, K. Gimpel, Gaussian error linear units (gelus).
- 688 [36] S. Elfving, E. Uchibe, K. Doya, Sigmoid-weighted linear units for neural
689 network function approximation in reinforcement learning 107 3–11, pub-
690 lisher: Elsevier.
- 691 [37] P. Ramachandran, B. Zoph, Q. V. Le, Swish: a self-gated activation func-
692 tion 7 1, publisher: Technical report.
- 693 [38] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-
694 parameter optimization 24.
- 695 [39] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-Tzur, M. Hardt,
696 B. Recht, A. Talwalkar, A system for massively parallel hyperparameter
697 tuning 2 230–246.
- 698 [40] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization.
- 699 [41] L. N. Smith, No more pesky learning rate guessing games 5.
- 700 [42] L. N. Smith, Cyclical learning rates for training neural networks, in: 2017
701 IEEE winter conference on applications of computer vision (WACV), IEEE,
702 pp. 464–472.
- 703 [43] L. N. Smith, N. Topin, Super-convergence: Very fast training of neural
704 networks using large learning rates, in: Artificial Intelligence and Machine
705 Learning for Multi-Domain Operations Applications, Vol. 11006, Interna-
706 tional Society for Optics and Photonics, p. 1100612.

- 707 [44] L. Prechelt, Early stopping-but when?, in: *Neural Networks: Tricks of the*
708 *trade*, Springer, pp. 55–69.
- 709 [45] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recog-
710 *nit*ion, in: *Proceedings of the IEEE conference on computer vision and*
711 *pattern recognition*, pp. 770–778.
- 712 [46] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye,
713 G. Anderson, G. Corrado, W. Chai, M. Ispir, others, Wide & deep learning
714 for recommender systems, in: *Proceedings of the 1st workshop on deep*
715 *learning for recommender systems*, pp. 7–10.
- 716 [47] S. G. Krantz, H. R. Parks, *The implicit function theorem: history, theory,*
717 *and applications*, Springer Science & Business Media.
- 718 [48] M. L. Michelsen, W. Yan, E. H. Stenby, A comparative study of reduced-
719 *variables-based flash and conventional flash*, *SPE Journal* 18 (05) (2013)
720 952–959.
- 721 [49] G. M. Kontogeorgis, G. K. Folas, *Thermodynamic models for industrial ap-*
722 *plications: from classical and advanced mixing rules to association theories,*
723 *John Wiley & Sons.*
- 724 [50] M. S. Wertheim, Fluids with highly directional attractive forces. II. ther-
725 *mod*ynamic perturbation theory and integral equations 35 (1) 35–47, pub-
726 *lisher: Springer.*
- 727 [51] M. Wertheim, Fluids with highly directional attractive forces. i. statistical
728 *thermodynamics* 35 (1) 19–34, publisher: Springer.
- 729 [52] M. Wertheim, Fluids with highly directional attractive forces. IV. equilib-
730 *rium polymerization* 42 (3) 477–492, publisher: Springer.
- 731 [53] M. Wertheim, Fluids with highly directional attractive forces. III. multiple
732 *attraction sites* 42 (3) 459–476, publisher: Springer.

- 733 [54] W. G. Chapman, K. E. Gubbins, G. Jackson, M. Radosz, New reference
734 equation of state for associating liquids 29 (8) 1709–1721, publisher: ACS
735 Publications.
- 736 [55] S. H. Huang, M. Radosz, Equation of state for small, large, polydisperse,
737 and associating molecules 29 (11) 2284–2294, publisher: ACS Publications.
- 738 [56] G. M. Kontogeorgis, E. C. Voutsas, I. V. Yakoumis, D. P. Tassios, An
739 equation of state for associating fluids 35 (11) 4310–4318, publisher: ACS
740 Publications.
- 741 [57] G. M. Kontogeorgis, I. V. Yakoumis, H. Meijer, E. Hendriks, T. Moorwood,
742 Multicomponent phase equilibrium calculations for water–methanol–alkane
743 mixtures 158 201–209, publisher: Elsevier.