A Molecular–Continuum Multiscale Model for Inviscid Liquid–Vapor Flow with Sharp Interfaces

Jim Magiera and Christian Rohde

University of Stuttgart, Institute of Applied Analysis and Numerical Simulation

Abstract

The dynamics of compressible liquid–vapor flow depends sensitively on the microscale behavior at the phase boundary. We consider a sharp-interface approach, and propose a multiscale model to describe liquid–vapor flow accurately, without imposing ad-hoc closure relations on the continuum scale. The multiscale model combines the Euler equations on the continuum scale with molecular-scale particle simulations that govern the interface motion. We rely on an interface-preserving moving mesh finite volume method to discretize the continuum-scale sharp-interface flow in a conservative manner. Computational efficiency, while preserving physical properties, is achieved by a surrogate solver for the interface dynamics based on constraint-aware neural networks.

The multiscale model is presented in its general form, and applied to regimes of temperature-dependent liquid–vapor flow which have not been accessible before.

1. Introduction

In two-phase flow the fluid properties vary strongly among their different phases. This makes modeling two-phase flow quite challenging, as for example on a numerical level the mass transfer for large density jumps has to be resolved accurately. Depending on the spatial scale, there are several approaches to model two-phase flow. Our focus lies on a spatial scale where all phase boundaries are explicitly represented. In this work, we consider a sharp-interface approach for ideal fluid flow governed by the Euler equations as a system of first-order conservation laws. Thus, the fluid domain $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, is split into two subdomains: the liquid-phase domain $\Omega_-$ and the vapor-phase domain $\Omega_+$, and both domains are connected via a codimension 1 phase boundary $\Gamma(t)$. In each of these domains only the respective phase is present, and both phases are coupled via the free interface using the Rankine–Hugoniot conditions. To determine the interface motion, further conditions have to be prescribed at the interface. Classically this is done by imposing an algebraic relation to fix the entropy dissipation rate [56]. However, determining the explicit form of this relation remains a largely unsolved issue for complex flow regimes. We model the interface flow dynamics on a more fundamental level and thus avoid prescribing some ad-hoc closure relations at the interface. Precisely, we simulate two-phase flow directly on the molecular scale using molecular dynamics (MD). This is attractive, because each molecule has the same properties and parameters regardless of the fluid phase it is part of. With this ansatz we develop a multiscale model that describes interfacial mass, momentum, and energy transfer, as well as the motion of the phase boundary by an MD interface solver.
We note that there is no rigorous mathematical theory that allows us to directly and analytically bridge the scales from MD systems with pairwise interactions to continuum-scale Euler equations. Yet, it is still possible to use classical, statistical averaging to infer continuum-scale quantities from microscale particle simulations. In that way, we transfer the microscale behavior at fluid interfaces to the continuum scale via a data-based approach. This approach can be categorized as a heterogeneous multiscale method (HMM) to couple atomistic- and continuum-scale models [16]. Moreover, we present a finite volume method on interface-preserving moving meshes (FV-IPMM-method) which enables us to include the molecular-scale dynamics directly at the phase boundary, while resolving the sharp interface directly within the mesh. The corresponding moving mesh-algorithms are presented in [5]. Finally, to overcome the computational costs of expensive MD simulations, we apply a surrogate for the MD interface solver. This surrogate is based on constraint-aware neural networks [37], to achieve computational efficiency, while preserving the mass-conservation of the numerical discretization.

Two-phase fluid flow models with sharp interfaces have already been widely investigated, see e.g. [17, 21, 34, 49] for various modeling approaches. Common discretization techniques involve for example cut-cell approaches [44, 51] and interface tracking using level-sets [19, 33]. Compared to most of these, our discretization method does not require interface reconstruction steps, nor an extra scalar field for phase-tracking.

MD simulations are a widely used tool in phase-transition applications. However, the integration into continuum-scale dynamics for compressible liquid–vapor flow has been rarely done. A direct comparison of MD simulations and continuum-scale liquid–vapor flow has been carried out in e.g. [20, 24]. In contrast to our contribution, the microscopic flow dynamics have not been directly integrated in the continuum-scale simulations over a whole range of flow regimes.

In recent years, a lot of attention was on the application of neural networks in the context of hyperbolic conservation laws. For example in [46] physics-informed neural networks were designed for conservation laws. In [7] neural networks are applied to infer reconstruction weights and dissipation terms for non-classical shock-waves. In our contribution, we focus on the application of neural networks as a surrogate solver, that assist in bridging the scales from the molecular to the continuum scale, while retaining important physical constraints such as mass conservation.

Outline First, we introduce in Section 2 a general sharp-interface two-phase model which is based on conservation laws. At the end of this section, we introduce a compressible liquid–vapor flow model, that serves as an explicit instance of the general model. In Section 3, we introduce the concept of a microscale interface solver that is used to describe the phase boundary dynamics. Thereafter, we present an interface solver that is based on MD simulations. For actual simulations, we do not employ the microscale interface solver directly, but instead use a surrogate solver that is described in Section 4. In Section 5, we provide the continuum-scale discretization scheme. It uses a moving-mesh method [5] to resolve the sharp-interface directly within the mesh. In Section 6, we present the multiscale model in a general form and provide a specific instance for the temperature-dependent two-phase flow model. In Section 7, we show numerical simulations results for this model. Finally, we discuss our findings and conclusions in Section 8.

The content of this contribution is part of the PhD thesis [35], preliminary parts are published in [39].
2. Continuum-Scale Sharp-Interface Model for Two-Phase Flow

We begin by introducing a general sharp-interface model for two-phase phase flow, which is based on first-order conservation laws.

2.1. The General Two-Phase Flow Model

Geometric Setting The central assumption for the sharp-interface approach is that the fluid domain \( \Omega \subset \mathbb{R}^d \), \( d \in \mathbb{N} \), is divisible into two disjoint, open and \( d \)-dimensional subdomains \( \Omega_-(t) \), \( \Omega_+(t) \), fulfilling \( (\Omega_-(t) \cup \Omega_+(t))' = \Omega \) for each \( t \geq 0 \). Here, each bulk domain \( \Omega_+(t) \) is filled with one respective fluid phase only, and the dividing hypersurface \( \Gamma(t) = \Omega_-(t) \cap \Omega_+(t) \) is called the (sharp) interface, see Figure 1 for a sketch of a typical setting. For a point \( \xi \in \Gamma(t) \) the normal vector pointing in direction of the + phase is denoted by \( n = n(\xi, t) \in \mathbb{S}^{d-1} \). The speed of the phase boundary \( \Gamma(t) \) in normal direction is denoted by \( s = s(\xi, t) \in \mathbb{R} \).

Bulk Equations We assume that in both bulk phases \( \Omega_+(t) \) the same set of first-order conservation laws are fulfilled, i.e.

\[
\begin{align*}
\partial_t U^+ + \nabla \cdot f(U^+) &= 0, & \text{in } \Omega_+(t) \text{ for } t \in (0, t_{\text{end}}), \\
U^+(\cdot, 0) &= U_0^+ & \text{in } \Omega_+(0),
\end{align*}
\]

with \( U^+ : \Omega_+(t) \times [0, t_{\text{end}}) \rightarrow \mathcal{H} \) being the unknown fluid states, with the open set \( \mathcal{H} \subset \mathbb{R}^m \), \( m \in \mathbb{N} \), denoting the state space of the system. The flux function is given by \( f \in C^1(\mathcal{H}, \mathbb{R}^{m \times d}) \), and the divergence operator in (1) is understood to act on each row of \( f(U^+) \). The initial conditions are given by \( U_0^+ : \Omega(t) \rightarrow \mathcal{H} \). For a more compact notation, we write

\[
U(x, t) := \begin{cases} 
U^+(x, t), & \text{if } x \in \Omega_+(t), t \in [0, t_{\text{end}}), \\
U^-(x, t), & \text{if } x \in \Omega_-(t), t \in [0, t_{\text{end}}). 
\end{cases}
\]

We define the flux in direction of an arbitrary \( n \in \mathbb{S}^{d-1} \) by

\[
F(U; n) := f(U) \cdot n.
\]

The system of conservation laws (1) is called hyperbolic, if for all \( U \in \mathcal{H} \) and \( n \in \mathbb{S}^{d-1} \) there exist \( m \) real eigenvalues \( \lambda_1(U; n), \ldots, \lambda_m(U; n) \), and \( m \) corresponding linear independent eigenvectors \( r_1(U; n), \ldots, r_m(U; n) \) of the flux Jacobian \( DF(U; n) : \mathcal{H} \rightarrow \mathbb{R}^{m \times m} \).
Interface Conditions  Let \( n = n(\xi, t) \) be the outer unit normal at \( \xi \in \Gamma(t) \). We define the jump operator \([\cdot]\) at the interface \( \Gamma \) for a sufficiently smooth function \( a : \mathbb{R}^d \rightarrow \mathbb{R} \) by
\[
[a] := \overline{a} - \underline{a}, \quad \text{with} \quad \overline{a}_ \varepsilon := \lim_{\varepsilon \to 0^+} a(\xi + \varepsilon n).
\]
To ensure conservation across discontinuities the Rankine–Hugoniot conditions
\[
s[U] = [F(U; n)]
\]
must be fulfilled [14]. Typically, for phase transition problems, further algebraic jump conditions are formulated. Those are classically formed by kinetic relations [56] of the form
\[
\mathcal{K}(U, \underline{U}) = 0,
\]
for a prescribed function \( \mathcal{K} : \mathcal{P}_- \times \mathcal{P}_+ \rightarrow \mathbb{R} \) that is usually associated with driving forces at the interface. Together with appropriate boundary conditions, the system (1), with the jump conditions (3) and (4), is a free boundary value problem for \( U^+ \) and the interface \( \Gamma \).

Phase States  Mathematically, the different phases are expressed as connected subsets in the state space \( \mathcal{U} \) for which the system (1) is hyperbolic. This, of course, depends heavily on the flux function \( f \) and the state space \( \mathcal{U} \) itself. For most real world fluids (as well as two-phase model fluids) the state space \( \mathcal{U} \) contains (at least) two disjoint domains \( \mathcal{P}_- \supset \mathcal{P}_+ \subset \mathcal{U} \), in such a way that for all \( U \in \mathcal{P}_- \cup \mathcal{P}_+ \) the system (1) is hyperbolic. Yet for mixed states \( U \in \mathcal{U} \setminus (\mathcal{P}_- \cup \mathcal{P}_+) \) the system can become elliptic. The lack of global hyperbolicity is a major challenge, that prohibits the usage of standard solvers. The domains \( \mathcal{P}_-, \mathcal{P}_+ \) form the state space for the two phases, which are marked by the subscripts \(-, +\). Accordingly, we will speak of the \(-\)phase and the \(+\)phase.

We will use the abbreviated notation \( \bar{\cdot} \) to indicate both phases (separately).

2.2. The Liquid–Vapor Flow Model

In this section we consider liquid–vapor flow as an explicit instance of the general two-phase flow model of Section 2.1.

Bulk Equations  We use the subscript \(-\) to refer to the liquid phase, and the subscript \(+\) to refer to the vapor phase. In order to model the flow in a domain \( \Omega \subset \mathbb{R}^d \) with a sharp interface \( \Gamma(t) \), we follow the geometric setting described in Section 2.1. We assume that the Euler equations hold in both bulk phases, i.e.
\[
\frac{\partial}{\partial t} \rho + \nabla \cdot (\rho \mathbf{v}) = 0,
\]
\[
\frac{\partial}{\partial t} (\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} + p \mathbf{I}) = 0, \quad \text{in } \Omega_\varepsilon(t) \text{ for } t \in (0, t_{\text{end}}).
\]
\[
\frac{\partial}{\partial t} E + \nabla \cdot ((E + p)\mathbf{v}) = 0,
\]
Again, we make use of the abbreviated notation (2). The unknowns of the system (5) are the fluid density \( \rho = \rho(x, t) \), the fluid momentum density \( \rho \mathbf{v} = \rho \mathbf{v}(x, t) \), the total energy density \( E = E(x, t) \), and the interface \( \Gamma(t) \). The total energy density satisfies \( E = \rho \varepsilon + \frac{1}{2} \rho \|\mathbf{v}\|^2 \), with \( \varepsilon \)
denoting the specific internal energy. The $d$-dimensional identity matrix is written as $I$. The flux function $f$ of (5) is given by

$$ f(U) := \begin{pmatrix} m^T \\ \frac{1}{2} m \otimes m + \rho I \\ (E + \rho)v^T \end{pmatrix}, $$

with the state vector $U := (\rho, m, E)^T$, where $m := \rho v$ denotes the momentum density.

To further close the system, the pressure $p$, the specific internal energy $\varepsilon$, and the temperature $T$ are expressed in terms of thermodynamically independent quantities by an equation of state (EOS). Many EOS do not prescribe e.g. the pressure directly. Instead, the pressure can be expressed via $p(\rho) = \rho^2 \frac{d}{d\rho} \psi(\rho)$, with $\psi$ being the specific Helmholtz free energy. Likewise, the specific internal energy can be computed via $\varepsilon(\rho, T) = \psi(\rho, T) - T \frac{d}{dT} \psi(\rho, T)$, as well as the internal entropy $S$ via $S = S(\rho, T) = -\partial_T \psi(\rho, T)$.

As discussed in [14], the system (5) is hyperbolic if the EOS fulfills

$$ \partial_\rho p(\rho, S) > 0, \quad T(\rho, S) > 0. $$

The admissibility criterion (6) divides the state space $\mathcal{U}$, depending on the EOS, into admissible regions that are identified with different fluid phases. We denote the domain in the state space $\mathcal{U}$ that corresponds to the liquid phase by $\mathcal{P}_-$, and for the vapor phase by $\mathcal{P}_+$.

**Equation of State for a Lennard–Jones Fluid** To describe the continuum-scale properties of the fluid in (5), we have to specify an explicit EOS. We choose the EOS presented in [54] that is consistent with a Lennard–Jones fluid on the molecular scale, and can be used to describe e.g. noble gases. A graphical representation of the pressure $p = p(\rho, T)$ in terms of density $\rho$ and temperature $T$ is shown in the phase diagram in Figure 2.

**Interface Conditions** The Rankine–Hugoniot conditions (3) for the system (5) write as

$$ \| \rho(\mathbf{v} \cdot \mathbf{n} - s) \| = 0, $$

$$ \| \rho(\mathbf{v} \cdot \mathbf{n} - s) \mathbf{v} \cdot \mathbf{n} + \mathbf{p} \| = 0, $$

$$ \| E(\mathbf{v} \cdot \mathbf{n} - s) + \rho \mathbf{v} \cdot \mathbf{n} \| = 0. $$

Figure 2.: Pressure and temperature of the Lennard–Jones EOS [54]. The dashed line represents the isobar of the critical pressure $p_{\text{crit}} = p(\rho_{\text{crit}}, T_{\text{crit}})$. 

(a) Pressure $p = p(\rho, T)$

(b) Temperature $T = T(\rho, \varepsilon)$
Note that we neglect interface curvature and surface tension effects. If these effects are taken into account, the right-hand sides of (7)_{2,3} are non-zero and depend on the local curvature.

In contrast to hydrodynamical shock waves, phase boundaries usually occur as subsonic waves, see Figure 3 for a typical Riemann wave pattern. Hence, additional criteria have to be imposed, to ensure a unique solution ([8, 56]). In classical theories [1, 30–32] this is done by prescribing kinetic relations like in (4). For example, one can specify a driving force \( \mathcal{K}: \mathbb{R} \rightarrow \mathbb{R} \) that describes the entropy production across the interface. In this instance, the kinetic relation is given by

\[
[S] - \mathcal{K}(j) = 0,
\]

with \( S = S(\rho, T) \) denoting the internal entropy of the fluid and \( j := \rho_s(v_s \cdot n - s) \) the relative mass flux.

### 3. The Microscale Interface Solver

The core problem of describing two-phase flow lies in the accurate description of the phase boundary dynamics. Within our framework, this can be realized by solving a planar two-phase Riemann problem, i.e. by solving (1) for the Riemann initial data

\[
U_0(x) = \begin{cases} 
U_{\text{L}} & \text{for } x < 0, \\
U_{\text{R}} & \text{for } x > 0,
\end{cases}
\]

with \( U_{\text{L}} \in \mathcal{P}_{\text{L}} \) constant, and \( x = (x - \xi) \cdot n \), for \( x \in \mathbb{R}^d \), \( \xi \in \Gamma \). Solving the two-phase Riemann problem analytically in the context of temperature-dependent liquid–vapor flow with sharp interfaces remains a largely unresolved task on the continuum-mechanical level. Despite the fact, that the two-phase Riemann problem has been widely analyzed in many contributions [1, 13, 22, 41, 52, 58], there remain major open issues for realistic EOS and temperature-dependent flow. One problem is that the physically relevant choice for the driving force \( \mathcal{K}(j) \) in the kinetic relation (8) is often unclear and depends on the type of fluid that is considered. Even more so, it is shown in [23, 53] that there are phase boundaries that fulfill the basic Rankine–Hugoniot jump conditions but contradict entropy constraints in the first-order framework.

These shortcomings of the continuum-scale model are overcome by our multiscale approach. In that we utilize molecular dynamics simulations at the interface. Thus, the driving forces are inherently included on the particle scale and do not have to be modeled separately on the continuum-scale. Before discussing the specific molecular-scale interface solver in detail, we start by introducing the general concept of a microscale interface solver.

#### 3.1. The Microscale Interface Solver for the General Two-Phase Framework

**Directional State Projections** The core problem of the multiscale model-algorithm is to find solutions of the local Riemann problems along the interface \( \Gamma(t) \). For this purpose, we regard the directional version of (1). Here, we have to distinguish between conserved quantities in \( U \) that are scalar (e.g. mass), and others that are vector-valued (e.g. momentum). Assume that \( U = (u_1, \ldots, u_k, v_1, \ldots, v_l) \in \mathbb{R}^m \) (after reordering), with \( u_i \in \mathbb{R}, i = 1, \ldots, k \), and \( v_i \in \mathbb{R}^d, i = 1, \ldots, l \), being...
primary unknowns of the system (1). For \( n \in \mathbb{S}^{d-1} \) the projected state in direction of \( n \) and its perpendicular remainder are defined by

\[
U_{\mid n} := (u_1, \ldots, u_k, v_1 \cdot n, \ldots, v_l \cdot n) \in \mathbb{R}^{k+l},
\]

\[
U_{\perp n} := (0, \ldots, 0, v_1 - (v_1 \cdot n)n, \ldots, v_l - (v_l \cdot n)n) \in \mathbb{R}^m.
\] (9)

The directional state \( U_{\mid n} \) can be transferred back to its original form via the operator

\[
P_n : \mathbb{R}^{k+l} \to \mathbb{R}^m : (u_1, \ldots, u_k, v_1, \ldots, v_l) \mapsto (u_1, \ldots, u_k, v_1 n, \ldots, v_l n).
\] (10)

Indeed, it holds \( U = P_n(U_{\mid n}) + U_{\perp n} \). For the projection in case of the liquid–vapor flow model we refer to (29).

**Rotated Riemann Problem in Normal Direction** \( n \in \mathbb{S}^{d-1} \). Let \( x = (x - \xi) \cdot n \), with \( x \in \mathbb{R}^d \), \( \xi \in \Gamma \). The rotated initial value problem corresponding to (1) is given by

\[
\partial_t u + \partial_x F(u; n) = 0, \quad \text{in } \mathbb{R} \times (0, t_{\text{end}}),
\]

\[
u(x, 0) = u_0(x, 0), \quad \text{in } \mathbb{R},
\] (11)

where \( u := U_{\mid n} \) is the projected state in direction of \( n \). For \( U_- \), \( U_+ \in \mathcal{U} \), we define the rotated Riemann initial data

\[
u_0(x) = \begin{cases} u_- & \text{for } x < 0, \\ u_+ & \text{for } x > 0, \end{cases}
\] (12)

where \( u_- := (U_-)_n \), \( u_+ := (U_+_n) \) are the projected states in direction of \( n \).

**Wave of Interest** On the continuum scale, the solution of the Riemann problem (11), (12) consists of a composition of elementary waves, such as shocks, contacts, and rarefactions. Out of them, we are interested in the dynamics of one specific, discontinuous wave of interest, which represents the sharp interface. We refer to Figure 3 for an illustration of a typical wave pattern. For the wave of interest we denote its two adjacent states by \( u'_-, u'_+ \in \mathbb{R}^{k+l} \), and the wave speed by \( s \in \mathbb{R} \). This corresponds to a traveling wave

\[
u'(x, t) = \begin{cases} u'_- & \text{for } x < st, \\ u'_+ & \text{for } x > st. \end{cases}
\] (13)
The traveling wave (13) is a weak solution of the rotated Riemann problem with the initial states $u^\epsilon, u^\epsilon$. As a prerequisite for weak solutions, the Rankine–Hugoniot conditions

$$F(u^\epsilon; n(\xi)) - F(u^\epsilon; n(\xi)) = s (u^\epsilon - u^\epsilon),$$

(14)

have to be fulfilled at the interface $\Gamma(t)$.

Projected back to the full-dimensional system, the wave of interest at the interface $\Gamma(t)$ consists of the states $U^\epsilon := P^\epsilon_n(u^\epsilon) + (U^\epsilon)_{\perp n} \in \mathcal{P}_+, \text{ and the wave speed } s \in \mathbb{R}$. The tuple $(U^\epsilon, U^\epsilon, s)$ forms a traveling wave

$$U^\prime(x, t; \xi) = \begin{cases} u^\epsilon & \text{for } x \cdot n(\xi) < st, \\ u^\epsilon & \text{for } x \cdot n(\xi) > st, \end{cases}$$

which is itself a weak solution of (1) with discontinuous initial data consisting of the two adjacent states.

**General Interface Solver** Our main interest lies in the construction of an interface solver, that computes the adjacent states at the interface, and the interface speed, for given initial data (12). We define such a solver in the following way.

**Definition 1** (General Interface Solver). Let $U_\epsilon \in \mathcal{P}_-, U^\epsilon \in \mathcal{P}_+$, and $n \in S^{d-1}$ be given. We assume that for the corresponding Riemann initial data (12) a unique weak solution of the rotated system (11) exists, and is composed of elementary waves that include a (discontinuous) wave of interest, defined by the adjacent states $u^\epsilon, u^\epsilon \in \mathbb{R}^{k+1}$ and wave speed $s \in \mathbb{R}$. Then, if the states $U^\epsilon := P^\epsilon_n(u^\epsilon) + (U^\epsilon)_{\perp n}$ are elements of $\mathcal{P}_+$, we call the mapping

$$\mathcal{R} : \mathcal{P}_- \times \mathcal{P}_+ \times S^{d-1} \rightarrow \mathcal{P}_- \times \mathcal{P}_+ \times \mathbb{R} : (U_\epsilon, U^\epsilon, n) \mapsto (U^\epsilon, U^\epsilon, s)$$

a **general interface solver** of the system (1).

As discussed in Section 2.2, such an interface solver is not easily available in a closed formulation as an exact solution of the Riemann problem.

**Remark 2.** For the construction of the interface solver, we are mostly interested in the dynamics of the Riemann problem (12) of the rotated system (11). If the flux function $f$ is rotationally invariant, or $d = 1$, it suffices to construct an interface solver of the form

$$R : \mathcal{P}_- \times \mathcal{P}_+ \rightarrow \mathcal{P}_- \times \mathcal{P}_+ \times \mathbb{R} : (u_\epsilon, u^\epsilon) = (u^\epsilon, u^\epsilon, s),$$

(15)

with $P_\epsilon$ being the corresponding projections of the phase state spaces $\mathcal{P}_\pm$.

### 3.2. The Molecular-Scale Interface Solver

One of the most detailed ways to describe fluid flow on a microscopic scale is to perform MD simulations, where each molecule is represented as a single particle. This has already been successfully applied to investigate liquid–vapor flow [25, 40], albeit, mainly focused on equilibrium computations.

In this section, first, we summarize some basics for MD simulations (see also [6] for a general introduction). For details on our specific MD simulations we refer to Appendix A. Then, we apply MD simulations to construct a novel microscale interface solver that describes the non-equilibrium dynamics of the phase boundary.
**MD System** For an MD particle system we consider $N_p \in \mathbb{N}$ particles with masses $m_i > 0$, positions $x_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, velocities $v_i \in \mathbb{R}^3$, and accelerations $a_i \in \mathbb{R}^3$, with $i = 1, \ldots, N_p$, inside a molecular-scale domain $\Omega \subset \mathbb{R}^3$. The particles interact via an inter-particle, two-body potential $\phi : (0, \infty) \to \mathbb{R}$. We consider the Lennard–Jones potential

$$\phi(r) = 4\varepsilon \left( \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^6 \right),$$

where $r > 0$ denotes the distance between two particles. The parameters $\sigma, \varepsilon > 0$ specify the type of particles and therefore the type of fluid. For $\sigma = 1$ and $\varepsilon = 1$ we get a Lennard–Jones fluid whose properties are described by the EOS [54] of Section 2.2. The force between two arbitrary particles $i \neq j$ computes as

$$f_{ij} = -\nabla_{x_i} \phi(|x_i - x_j|).$$

With this, we can formulate the equations of motion for each particle $i = 1, \ldots, N_p$ and time $\tau \geq 0$ by

$$\frac{d}{d\tau} x_i(\tau) = v_i(\tau), \quad \frac{d}{d\tau} v_i(\tau) = a_i(\tau), \quad a_i(\tau) = \sum_{j \neq i} f_{ij}(\tau)/m_i,$$

subject to the initial conditions

$$x_i(0) = x_i^0, \quad v_i(0) = v_i^0.$$

Note that the MD simulations are always three-dimensional, as only in this case the micro-scale fluid is consistent with the continuum-scale fluid and its EOS. The discretization of the particle system (17) is given in Algorithm 10. By using the CUDA framework [43] we are able to utilize graphics processing units (GPUs) that achieve a high computational performance for such tasks.

**Molecular-Scale Riemann Problem** Our main interest lies in computing Riemann solutions on the molecular scale for the continuum-scale initial conditions $u_\ast = (\rho_\ast, v_\ast, T_\ast)$ in the liquid phase and $u_\ast = (\rho_\ast, v_\ast, T_\ast)$ in the vapor phase, as in (12). Both phases are divided by a molecular-scale interface which we consider as a plane positioned at $\Gamma(0) \in \mathbb{R}$ in $x$-direction.

To set up such a molecular-scale Riemann problem, we choose a fixed number of particles $N_p \in \mathbb{N}$ and distribute them into two subdomains with fixed width in transversal directions, such that the desired density is achieved and the ratio of the subdomain lengths in $x$-direction is fixed. In the next step, both particle systems have to undergo a thermalization phase, to attain an appropriate thermodynamic equilibrium state corresponding to $u_\ast$. In this phase, we consider each subsystem separately with periodic boundary conditions. Then, we run the MD simulation for a fixed number of time steps, while applying a thermostat in each subdomain, to attain the desired temperatures $T_\ast$. Afterwards, the bulk velocities $v_\ast$ are added to all particle velocities in each respective subdomain. This realizes the continuum-scale fluid velocities on the particle scale.

Finally, both domains are combined by putting them next to another, while retaining a small gap $l_{\text{gap}} = 2\pi\sigma$ in between, in order to avoid overlapping particles. In that way, we have set up the initial particle system. The whole initialization procedure is described in Algorithm 11 in Appendix A — for an illustration we refer to Figure 4.
Figure 4.: Sketch of the initialization of a molecular-scale Riemann problem.
**Interface Tracking** To get the position of the liquid–vapor interface, we exploit the fact that there is a large density jump from the liquid to the vapor phase. The first step is to transfer the particle distribution \((x_i(\tau))_{i=1,\ldots,N_p}\) to a continuous density distribution \(\hat{\rho}(x, \tau)\) along the \(x\)-axis by employing radial basis functions (RBFs)

\[
\hat{\rho}(x, \tau) := \sqrt{\frac{\gamma_{\text{RBF}}}{\pi}} \sum_{i=0}^{N_p} m_i \exp\left(-\gamma_{\text{RBF}}(x - x_i(\tau))^2\right),
\]

where \(\gamma_{\text{RBF}} > 0\) is the RBF-parameter, controlling the smoothing. The derivative with respect to \(x\) computes as

\[
\partial_x \hat{\rho}(x, \tau) = -2 \sqrt{\frac{\gamma_{\text{RBF}}}{\pi}} \sum_{i=0}^{N_p} m_i (x - x_i(\tau)) \exp\left(-\gamma_{\text{RBF}}(x - x_i(\tau))^2\right).
\]

During the MD simulation (computed using the time-step Algorithm 10) we compute the interface position at every time step \(\tau_n\), by finding the maximum of \(\partial_x \hat{\rho}(x, \tau_n)\), yielding the new interface position \(\Gamma(\tau_n) \in \mathbb{R}\). For the local optimization we use basic optimization routines implemented in e.g. [28].

After obtaining the interface position \(\Gamma(\tau_n)\), we infer the interface speed by performing a simple linear regression over the last \(n_{\text{buffer}} \in \mathbb{N}\) interface positions \((\tau_j, \Gamma(\tau_j)), j = \max(n - n_{\text{buffer}}, 0), \ldots, n\). The slope of the fitted line provides the approximation for the interface speed \(s = s(\tau_n)\) at time \(\tau_n\). The whole procedure is described in Algorithm 12 in Appendix A.

**Interface State Sampling** For the multiscale model, it is necessary to compute the fluid states \(\mathbf{u}_s = (\rho^s, \mathbf{m}^s, T_s)\) adjacent to the interface, see Figure 3. To compute these states during the MD simulation, we define a sampling region \(\Sigma_s(\tau)\) near the interface for each phase:

\[
\Sigma_s(\tau) := [\Gamma(\tau) - \omega_s - o_s, \Gamma(\tau) - o_s], \quad \Sigma_s(\tau) := [\Gamma(\tau) + o_s, \Gamma(\tau) + \omega_s + o_s],
\]

where \(\omega_s > 0\) denotes the width of the sampling region, and \(o_s \geq 0\) the interface offset. The width of the sampling region is determined heuristically, but it has a large impact on the simulation results. Making it too small results in high statistical errors as the number of particles in the sampling domain decreases. In contrast, making it too large may give incorrect wave states if other waves are included. This means the sampling region should not be larger than the wave state plateau at the interface.

The interface offset accounts for the small transition region near the interface, which should be avoided in the averaging procedure.

Having defined the sampling regions, we can compute the interface wave states by employing the Irving–Kirkwood formulas [15, 26]

\[
\rho^s_s(\tau) = \text{vol}(\Sigma_s)^{-1} \sum_{x_i(\tau) \in \Sigma_s} m_i, \\
\mathbf{m}^s_s(\tau) = \text{vol}(\Sigma_s)^{-1} \sum_{x_i(\tau) \in \Sigma_s} m_i \mathbf{v}_i(\tau), \\
\mathbf{v}^s_s(\tau) = \left( \sum_{x_i(\tau) \in \Sigma_s} m_i \right)^{-1} \sum_{x_i(\tau) \in \Sigma_s} m_i \mathbf{v}_i(\tau), \\
T^s_s(\tau) = |x_i(\tau) \in \Sigma_s|^{-1} \frac{1}{d} \sum_{x_i(\tau) \in \Sigma_s} m_i (\mathbf{v}_i(\tau) - \mathbf{v}^s_s(\tau)) \cdot (\mathbf{v}_i(\tau) - \mathbf{v}^s_s(\tau)), \tag{18}
\]

where \(d\) is the dimension of the phase space.
where \( \text{vol}(\Sigma_+) = w_r h_{\text{dom}}^2 \) denotes the volume of the sampling region, and \( |\chi_i(r) \in \Sigma_+| \) the number of particles inside it. In that way, we obtain the local interface states at every point in time during a MD simulation. To get even more reliable results, we perform time averaging over a fraction \( \alpha_{t\text{-sampl}} \in [0, 1] \) of the total simulation time \( \tau_{\text{end}} > 0 \).

A graphical illustration of the interface state sampling and the interface tracking procedure is shown in Figure 5.

**Molecular-Scale Interface Solver** By applying the methods described in the previous sections, we are able to initialize a molecular-scale Riemann problem, run MD simulations, and compute the interface dynamics. The whole procedure results in an interface solver (of the same form as (15))

\[
R_{\text{MD}}(\rho_-, \mathbf{v}_-, T_-, \rho_+, \mathbf{v}_+, T_+) = (\rho'_- , \mathbf{v}'_-, T'_-, \rho'_+ , \mathbf{v}'_+, T'_+, s).
\]

It maps the Riemann initial data to the interface wave states and the interface speed. The algorithmic description of (19) follows.
Algorithm 3: Molecular-Scale Interface Solver

**Input:** continuum fluid states \((\rho_-, \nu_-, T_-), (\rho_+, \nu_+, T_+)\).

**Parameters:** total number of particles \(N_p\), processing frequency \(f_{pr} \in \mathbb{N}\), number of time steps \(N_{end}\), time step \(\Delta t > 0\), time sampling ratio \(\alpha_{-\text{smpl}} \in [0, 1]\), sampling region width \(w_{sr} > 0\), sampling region interface offset \(o_{sr} \geq 0\), thermostat region offset ratio \(\alpha_{\text{therm}} \in [0, 1]\).

**Algorithm**

- Set up initial particle configuration of the molecular-scale Riemann problem according to the continuum fluid states \((\rho_-, \nu_-, T_-), (\rho_+, \nu_+, T_+)\) (see Algorithm 1).

- For \(n = 1, \ldots, N_{end}\):  
  - Compute new time steps according to Algorithm 10.
  - Compute new interface position \(\Gamma_n\) and interface speed \(s_n\) according to Algorithm 12.
  - Compute interface states \((\rho^*_-, \nu^*_-, T^*_-)\) as described in (18).

- Compute the time averaged values \((\rho^*_-, \nu^*_-, T^*_-, s)\) from \((\rho^*_-, \nu^*_-, T^*_-)\) for all \(n \in \{\tilde{a}_{-\text{smpl}}N_{end}, \ldots, N_{end}\}\).

**Result:** interface states \((\rho^*_-, \nu^*_-, T^*_-, s)\), \((\rho^*_+, \nu^*_+, T^*_+)\), and interface speed \(s\).

4. Surrogate Microscale Interface Solver

In most cases, microscale simulations or the computation of the interface dynamics in general are computationally expensive. To address this problem, we employ surrogate solvers that rely on machine learning algorithms, such as neural networks. The surrogate solver is denoted by

\[
R_\theta : P_- \times P_+ \rightarrow P_- \times P_+ \times \mathbb{R} : (u_-, u_+) \mapsto (\bar{u}^-_-, \bar{u}^+_+, s),
\]

where \(\theta\) constitutes the parameters of the surrogate model, which are determined by the training/optimization procedure. For neural networks these parameters consist of the weights and biases of the network.

By using such a surrogate solver we can reduce the computational time from a couple of minutes for a full microscale simulation to a fraction of milliseconds. This comes at the costs of generating a data set and training the surrogate solver. Nonetheless, applying surrogate solvers usually pays off after several evaluations, compared to more expensive microscale interface solvers \(R\). For a more detailed discussion of the computational gains we refer to Section 7.3.

**Constraint-Aware Neural Networks**  
Most traditional methods for generating a surrogate solver \(R_\theta\) do not incorporate physical constraints, such as the Rankine–Hugoniot conditions (14) at the interface. This is primarily due to the fact that most methods merely focus on fitting the data, without trying to incorporate underlying physical constraints. Additionally, the microscale solver itself might not fulfill such constraints. This may be due to dissipative effects that appear on the microscale (but not in the continuum-scale model). Furthermore, statistical noise on the
output values can invalidate constraints, which is the case for sampling from MD simulations. We mitigate this problem by employing constraint-aware neural networks, see [37], as our surrogate solver. They are able to resolve physical constraints exactly by adding a constraint-resolving layer to a neural network, which implements a constraint-resolving function $\Psi(z) = y$. To this end, we define the output variables $z$ of the previous part of the neural network, i.e. $G_b(x) = z$, and the mapping $\Psi(z) = y$, such that $y$ fulfills the physical constraints. The explicit form of the constraint-resolving function $\Psi$ depends on the specific model. A particular choice for $\Psi$ is given in (21) for liquid–vapor flow.

**Surrogate Solver Generation** In this section, the generation of the surrogate solver is outlined. For an in-depth description of the employed machine learning algorithm we refer to [37].

**Data Set Generation** To generate a surrogate solver, we need a data set $D := \{(x_i, y_i) : i = 1, ..., N_{data}\}$ describing the input–response-relation of the microscale interface solver we intend to substitute. Here, the input $x_i := (u_-, u_+)i$ is a pair of Riemann initial data, and $y_i := (u_-', u_+', s)_i$ constitutes the corresponding output of the microscale interface solver $R$, i.e. $y_i = R(x_i)$. Therefore, the first step is to generate an input data set $D_{in} = \{(u_-, u_+)i : i = 1, ..., N_{data}\}$. Without active/online sampling methods (see Remark 9), this step involves choosing a sampling domain $B_{in} := B_+ \times B_-, \subset \mathbb{R} \times \mathbb{R}$, that serves as the bounding domain for the input data, which means that all input points $x_i$ will lie in $B_{in}$. This is a crucial step, as the choice dictates for what kind of flow regimes the surrogate solver is valid. After determining such a sampling domain, input points for the microscale simulations are sampled in $B_{in}$ using the sampling Algorithm 14. This results in the input data set $D_{in}$.

Afterwards, each input point $x_i = (u_-, u_+)i \in D_{in}$ is evaluated by the microscale interface solver, i.e.

$$R(x_i) = y_i = (u_-', u_+', s)_i,$$

resulting in the output states $u_-', u_+',$ and the interface speed $s \in \mathbb{R}$. This yields the corresponding output data set $D_{out} = \{(u_-', u_+', s)_i : i = 1, ..., N_{data}\}$. Combining both data sets yields the complete data set $D := \{(x_i, y_i) : i = 1, ..., N_{data}\}$ that describes the input–response-relation of the microscale interface solver.

A large enough number of samples $N_{data}$ for the data set $D$ is necessary in order to apply the surrogate solver successfully. Note, that in this work we choose the data set size $N_{data}$ purely from an empirical standpoint. Of course, the size is not the only quality measure of a data set, the distribution of the input variables $x_i$ and inherent noise of the output variables $y_i$ play a role just as important. In order to achieve a well-spaced input distribution, we apply the sampling Algorithm 14. The output noise can be reduced by performing several experiments for each input sample point $x_i$ and averaging the outputs, resulting in an output label $y_i$.

### 4.1. Surrogate Solver for Liquid–Vapor Flow

Instead of running the whole MD simulation (Algorithm 3) for evaluating $R_{MD}$ in (15) we employ a surrogate solver. To decrease the dimensionality of the input space of the surrogate solver, we exploit the fact that as a particle system MD are invariant with respect to a moving reference
frame. Thus, we choose the --phase velocity \( u_\ast = \frac{m_\ast}{\rho_\ast} \) as the reference velocity \( u_{\text{ref}} := u_\ast \) and consider the fluid velocities \( \tilde{v}_\ast = v_\ast - u_{\text{ref}} \). Consequently, the surrogate solver takes the form
\[
R_\theta(x) = R_\theta(\rho_\ast, T_\ast, \rho_\ast, v_\ast, T_\ast) = R_{\text{MD}}(\rho_\ast, 0, T_\ast, \rho_\ast, v_\ast, T_\ast) = (\rho_\ast^\prime, m_\ast^\prime, T_\ast^\prime, \rho_\ast, m_\ast, T_\ast).
\]
The underlying data set, as well as some remarks regarding the training of the network are found in Appendix C.

As discussed in Section 4, generic surrogate solvers \( R_\theta \) are not tailored to uphold the Rankine–Hugoniot condition (7), which include e.g. mass balance across the phase boundary. We aim to include mass conservation and positivity of the density in the surrogate solver \( R_\theta \). For this purpose, we employ CRes-networks [37] as our surrogate solver. To build the necessary constraint-resolving function \( \Psi \), consider at first the mass conservation across the interface (7), rewritten as
\[
m_\ast = m_\ast - s(\rho_\ast - \rho_\ast) \tag{20}
\]
This identity is used in the constraint-resolving function \( \Psi \) to account for the mass conservation across the interface. Furthermore, we ensure that the returned densities remain positive.

The CRes-network consists of an intermediate network \( G_\theta(x) = z \) with the output
\[
z := (\tilde{\rho}_\ast, \tilde{\rho}_\ast, \tilde{T}_\ast, \tilde{T}_\ast, s),
\]
where \( \tilde{\rho}_\ast \) are the intermediate densities, \( \tilde{T}_\ast \) the intermediate phase temperatures, \( v_\ast^\prime \) the fluid velocity of the --phase, and \( s \) the interface speed. Then, we define the function \( \Psi \) as
\[
\Psi(\tilde{\rho}_\ast, v_\ast^\prime, \tilde{T}_\ast, \tilde{T}_\ast, s) := (|\tilde{\rho}_\ast|, |\tilde{\rho}_\ast|^\prime, |\tilde{T}_\ast|, |\tilde{T}_\ast|^\prime, |\tilde{\rho}_\ast|^\prime - s|\tilde{\rho}_\ast| - |\tilde{\rho}_\ast|, |\tilde{T}_\ast|^\prime, s) \tag{21}
\]
This choice ensures that the output densities \( \rho_\ast^\prime \) and temperatures \( T_\ast^\prime \) are positive, and by means of (20) the mass across the interface is conserved. The general structure of the corresponding CRes-network is shown in Figure 6.

In the end, a single evaluation of the complete surrogate solver takes less than a millisecond whereas a single evaluation of \( R_{\text{MD}} \) takes between 9 and 11 min. A more detailed discussion of the computational times is found in Section 7.3.

5. Finite Volume Method on Interface-Preserving Moving Meshes

The focus of this section lies on the numerical discretization of two-phase flow models of the form (1) with a sharp phase boundary. This method is utilized as the continuum-scale discretization for our multiscale model. For this purpose, we employ a finite volume method (FV-IPMM-method) that builds upon an interface-preserving moving mesh (IPMM) introduced in [5, 35]. In this contribution, we will use the IPMM only as a black box without going into details. In summary, the IPMM provides a simplicial conforming moving mesh that is able to preserve a discretized interface \( \Gamma(x,t) \). This means that the interface always coincides with cell surfaces. Furthermore, it is possible to prescribe the interface motion, while employing local remeshing (which is described in [5]). In that way, large mesh deformations are possible, in contrast to e.g. basic Lagrangian approaches. The main benefit of the FV-IPMM-method is, that there is no interpolation across
the interface, and that the interface dynamics can be directly described by an interface solver. A graphical illustration of the two-dimensional IPMM is shown in Figure 7. In the following, assume we have a conforming, time-dependent, IPMM $\mathcal{F}(t)$, $t \geq 0$, of the two-phase domain $\Omega(t) = \Omega_+(t) \cup \Omega_-(t) \cup \Gamma(t) \subset \mathbb{R}^d$. Note that the evolution of the interface — and consequently the moving mesh $\mathcal{F}(t)$ — is not known a priori, but is a result from the interface dynamics during the simulation.

We introduce some further notation. The set of vertices incident to the interface $\Gamma_+(t)$ is denoted by $\mathcal{V}_+(t)$, and the set of interface surfaces by $\mathcal{S}_+(t)$. The cell surfaces that lie inside the bulk phases are given by $\mathcal{S}_\setminus_+(t) = \mathcal{S}_+(t) \setminus \mathcal{S}_+(t)$, where $\mathcal{S}_+(t)$ denotes the set of all cell surfaces. The index set for all vertices $p \in \mathcal{V}(t)$ is denoted by $\mathcal{I}_V(t) \subset \mathbb{N}$, and the one for all cells in the mesh $\mathcal{F}(t)$ by $\mathcal{I}_C(t) \subset \mathbb{N}$.

**Numerical Flux** The FV-IPMM-method is based on a finite-volume scheme. As such a numerical flux function

$$G : \mathcal{U} \times \mathcal{U} \times \mathbb{S}^{d-1} \rightarrow \mathbb{R}^m$$

is needed. For the sake of simplicity, we employ only the Lax–Friedrichs flux function

$$G(U, V, n) := \frac{F(U; n) + F(V; n)}{2} - \alpha_{LF}(U - V),$$
with the parameter \( \alpha_i > 0 \), but it is possible to apply other numerical flux functions that can deal with EOS like [54].

**The FV-IPMM-Method**  We start by giving a rough outline of the derivation of the FV-IPMM-method. For this purpose, we integrate (1) over a cell \( C(t) \in \mathcal{T}(t) \) with time dependent vertices, and the time interval \( t \in [t_n, t_{n+1}] \). Assuming sufficient regularity, this results in applying Gauss’s theorem in

\[
\int_{t_n}^{t_{n+1}} \int_{C(t)} \partial_t U \ dx \ dt + \int_{t_n}^{t_{n+1}} \int_{\partial C(t)} f(U) \cdot n \ dS \ dt = 0.
\]

By applying a version of Reynolds’ theorem, we arrive at the cell-wise conservative form of (1), i.e.

\[
\int_{C(t_{n+1})} U(x, t_{n+1}) \ dx = \int_{C(t_n)} U(x, t_n) \ dx - \int_{t_n}^{t_{n+1}} \int_{\partial C(t)} (f(U) - U s^T) \cdot n \ dS \ dt,
\]

where \( s : \partial C(t) \to \mathbb{R}^d \) denotes the speed of the surface at a point \( \xi \in \partial C(t) \). For a thorough derivation we refer to [11]. The FV-IPMM-method is basically a discretization of the terms in (22), while distinguishing between cell surfaces that form the interface \( \Gamma(t) \), and the bulk phase surfaces. The algorithmic description follows.

**Algorithm 4: Finite Volume Method on Interface-Preserving Moving Meshes**

**Input:** initial mesh \( \mathcal{T}(0) \), cell-averaged initial data \( (U_i^0)_{i \in \mathcal{P}(0)} \), interface solver \( \mathcal{R} : \mathcal{P} \times \mathcal{P} \times \mathbb{R}^d \to \mathcal{P} \times \mathcal{P} \times \mathbb{R} \).

**Parameters:** time step \( \Delta t_n > 0 \), vertex motion regularization parameter \( \lambda_{\text{motion}} \geq 0 \), numerical flux function \( G : \mathcal{H} \times \mathcal{H} \times \mathbb{R}^d \to \mathbb{R}^m \).

**Algorithm**

For each time step \( t_n \to t_{n+1} \), do

- Initialize \( \Delta U_i = 0 \) for all \( i \in \mathcal{P}(t_n) \).
- For all interface surfaces \( S_{ij}^n \in \mathcal{P}(t_n) \) — with their incident cells being \( C_i, C_j \) and assuming without loss of generality \( U_i^n =: U_i \in \mathcal{H}_i, U_j^n =: U_j \in \mathcal{H}_j \) — do
  - Solve the interface Riemann problem
    \[
    (U_i^s, U_j^s, s) = \mathcal{R}(U_i, U_j, n_{ij}).
    \]
  - Add contributions to the finite volume update terms
    \[
    \Delta U_i = -\frac{\Delta t_n \ |S_{ij}^n|}{|C_i(t_n)|} \left( f(U_i^s) - s U_i^s \right), \quad \Delta U_j = -\frac{\Delta t_n |S_{ij}^n|}{|C_j(t_n)|} \left( f(U_j^s) - s U_j^s \right).
    \]
  - Save \((s, n_{ij}, |S_{ij}|)\) for every vertex \( p_k \) incident to \( S_{ij} \).
- Update vertex motion \( m_k \) for all \( k \in \mathcal{P}(t_n) \), see (26).
- For every surface inside the bulk domain \( S_{ij}^n \in \mathcal{S}_k(t_n) \)
Compute the linear motion contribution
\[ \ell_{ij} := \frac{1}{2} \left( |S_{ij}^{n+1}| + |S_{ij}^n| \right) \cdot \left( \frac{U_i^n + U_j^n}{2} + \frac{n_{ij}^{n+1} + n_{ij}^n}{2} \right) \cdot \frac{\sum_{p \in \mathcal{Y}(S_{ij}^n) m_k}{|\mathcal{Y}(S_{ij}^n)|}}, \]  
(23)

with \( \mathcal{Y}(S_{ij}^n) \) denoting the set of vertices \( p \in \mathcal{Y}(t_n) \) that form the cell surface \( S_{ij}^n \). Consequently, \( \ell_{ij} = 0 \) if \( S_{ij} \) is not incident to an interface vertex.

- Add contributions to the finite volume update terms
\[ \Delta U_i = -\frac{\Delta t_n}{|C_i(t_n)|} \left( |S_{ij}^n| G(U_i^n, U_j^n, n_{ij}^n) + \ell_{ij} \right), \]
\[ \Delta U_j = +\frac{\Delta t_n}{|C_j(t_n)|} \left( |S_{ij}^n| G(U_i^n, U_j^n, n_{ij}^n) + \ell_{ij} \right). \]

- Add update terms to all cells:
\[ U_i^{n+1} = U_i^n + \Delta U_i, \quad \text{for } i \in \mathcal{W}(t_n). \]

- Update vertex positions according to
\[ p_k^{n+1} = p_k^n + \Delta t_n m_k, \quad \text{for } k \in \mathcal{Y}(t_n), \]

and rescale the data of all neighboring cells according to the changing volume, i.e.
\[ U_i \leftarrow \frac{|C_i(t_n)|}{|C_i(t_{n+1})|} U_i. \]

- Perform the remeshing according to the IPMM (see [5]).

**Remark 5.** Concerning the FV-IPMM-method in Algorithm 4, there are some remarks to be made.

- The finite volume update terms \( \Delta U_i \) are used as variables, with \( +=, -= \) denoting assignment by addition or subtraction.

- By scaling the data in (24) the interface motion remains conservative. Even more so, we avoid mixing phase states in the context of two-phase flows.

- The linear motion contribution (23) originates from discretizing the motion-dependent part of the flux term in (22) for cell surfaces adjacent to the interface.

It remains to discuss how the interface motion is computed.

**Computing the Vertex Motion of the Mesh** In the FV-IPMM-scheme (Algorithm 4) the interface speed \( s \) is always computed for each cell surface \( S \in \mathcal{S}(t) \) in direction of its normal vector \( n \) — however, for the IPMM we require the motion at each vertex. As such we have to compute the vertex motion \( m_k = m_k(t) \in \mathbb{R}^d \) for each interface vertex \( p_k \in \mathcal{Y}(t) \). Let us assume that for each interface surface \( S_i \in \mathcal{S}(t) \) incident to \( p_k \) we have already computed the interface wave speed \( s_i \) in normal direction \( n_i \). In the multiscale model, this will be provided by the interface solver.
A natural choice for the interface motion \( \mathbf{m}_k \) of \( \mathbf{p}_k \) is one that satisfies

\[
\mathbf{n}_l \cdot \mathbf{m}_k = s_l \text{ for all surfaces } S_l \text{ incident to } \mathbf{p}_k,
\]

which means that the projection of \( \mathbf{m}_k \) in each normal direction \( \mathbf{n}_l \) coincides with the interface speed \( s_l \).

In one space dimension this is trivially fulfilled by setting \( \mathbf{m}_k = s \). In case of two space dimension, one vertex has usually (in our setting) only two interface edges, i.e. (25) boils down to a \( 2 \times 2 \) system of linear equations, that is solvable if the interface edges are not parallel. In three space dimensions each interface vertex has in most cases more than three incident interface surfaces, consequently, the linear system (25) is generally overdetermined. To solve this problem, linear least squares fitting, using the normal equations, can be performed. For that, we define

\[
\mathbf{N} := \begin{bmatrix} \mathbf{n}_1^\top \\ \vdots \\ \mathbf{n}_K^\top \end{bmatrix}, \quad \mathbf{s} := \begin{bmatrix} s_1 \\ \vdots \\ s_K \end{bmatrix},
\]

where \( \mathbf{n}_l^\top, s_l \) correspond to the normal vectors and surface speeds for all \( K \in \mathbb{N} \) incident surfaces \( S_l \) of \( \mathbf{p}_k \). Then the normal equations are given by

\[
\mathbf{N}^\top \mathbf{N} \mathbf{m}_k = \mathbf{N}^\top \mathbf{s}.
\]

To regularize this system, we apply the generalized Tikhonov regularization [55], with the expected value \( \mathbf{m}_k \) being the averaged motion

\[
\overline{\mathbf{m}}_k := \left( \frac{1}{|\delta_I(\mathbf{p}_k)|} \sum_{S_l \in \delta_I(\mathbf{p}_k)} s_l \right) \overline{\mathbf{n}}_k / |\overline{\mathbf{n}}_k|,
\]

with \( \delta_I(\mathbf{p}_k) \) denoting the interface surfaces incident to the vertex \( \mathbf{p}_k \), and the averaged normal vector given by

\[
\overline{\mathbf{n}}_k := \frac{1}{|\delta_I(\mathbf{p}_k)|} \sum_{S_l \in \delta_I(\mathbf{p}_k)} \mathbf{n}_l.
\]

Then, the motion \( \mathbf{m}_k \) of the vertex \( \mathbf{p}_k \) is the solution of the linear system

\[
(\mathbf{N}^\top \mathbf{N} + \lambda_{\text{motion}} \mathbf{I}) \mathbf{m}_k = \mathbf{N}^\top \mathbf{s} + \lambda_{\text{motion}} \overline{\mathbf{m}}_k,
\]

where \( \lambda_{\text{motion}} > 0 \) is the regularization parameter, and \( \mathbf{I} \) the \((d \times d)\)-identity matrix.

\textit{Remark 6.} It is noteworthy that the FV-IPMM-method is applicable beyond the multiscale model presented in this work. Many models can be applied in the bulk domains or the interface, and the method is not restricted to phase boundary scenarios. An open-source implementation of the IPMM can be found at [4].

6. The Multiscale Model

In this core section we present the multiscale model and the interplay of each of its components. We first describe it on the general level of Section 2.1. Thereafter, we will go into details and explain the specific implementation for liquid–vapor flow (Section 2.2), where the interface dynamics are described by MD simulations.
6.1. The Multiscale Model for General Two-Phase Flow

The multiscale model consists of several components. On the continuum scale, two-phase fluid flow can be modeled by conservation laws with sharp phase boundaries, see Section 2. Models of this type can be discretized via the FV-IPMM-method described in Section 5. A major benefit of this approach is, that it permits us to model the fluid dynamics directly at the moving interface, for example, by prescribing a general interface solver $R$ (see Definition 1). Following this idea, we model the interface dynamics with a multiscale model that is based on microscale simulations of the phase boundary, forming the interface solver. Such a microscale interface solver $R$ has the form given in (15). For given initial states $u_-$, $u_+ \in P_-$, $u_\ast \in P_+$, it solves the microscale Riemann problem (in normal direction) and extracts the wave speed $s \in \mathbb{R}$ of the interface as well as the adjacent wave states $u_-^\ast \in P_-$, $u_+^\ast \in P_+$ — see Figure 3 for an exemplary sketch of corresponding wave pattern. The microscale interface solver $R$ can be used to formulate a general interface solver $\mathcal{R}$ that is used in the FV-IPMM-method.

Specifically, for liquid–vapor flow, the microscale interface solver $R$ consists of MD simulations of molecular-scale Riemann problems. From these particle simulations the phase boundary dynamics are inferred. The MD interface solver is described in Section 3.2.

A graphical representation of the interplay between the different components of the multiscale model is shown in Figure 8.

**Remark 7.** The multiscale model is not restricted to employ particle-based microscale solvers for the general interface solver $\mathcal{R}$. There are many other possibilities that can be used. For example one could employ exact Riemann solvers [12, 50, 59], or (computationally cheaper) relaxation solvers [18]. The interface solver may also be based on other microscale models like diffuse-interface approximations [8] or completely different modeling approaches.
6.1.1. The Multiscale Model Algorithm

In this section, the application of the multiscale model is explained. It employs the surrogate interface solver $R_\partial$ in the FV-IPMM-discretization (Section 5) of a continuum-scale two-phase model.

We start with the initial data on the continuum scale, which means, we prescribe the initial bulk phase domains $\Omega_\pm(0)$, the initial interface position $\Gamma(0)$, and the corresponding initial conditions $U_{\pm,0}(x)$. Both domains $\Omega_\pm(0)$ are triangulated while approximating the interface $\Gamma(0)$. Then, we employ the FV-IPMM-method (see Algorithm 4). In the following, the multiscale model is given in algorithmic form.

**Algorithm 8: Multiscale Model Algorithm**

**Input:** initial bulk domains $\Omega_\pm(0)$, initial interface position $\Gamma(0)$, and initial conditions $U_{\pm,0}(x)$.

**Prerequisite:** a surrogate solver $R_\partial$, that is based on the microscale interface solver $\mathcal{R}$.

**Algorithm**

- Approximate $\Omega_\pm(0)$ and interface $\Gamma(0)$ by mesh with interface $\Gamma_{\mathcal{T}}$, comprising the initial moving mesh $\mathcal{T}(0)$.
- Compute cell-averaged data $(U_i^0)_{i \in I_{\pm}(0)}$ on $\mathcal{T}(0)$, using the initial conditions $U_{\pm,0}(x)$.
- Set up the interface solver

$$
\mathcal{R} : \mathcal{P}_- \times \mathcal{P}_+ \times \mathbb{S}^{d-1} \rightarrow \mathcal{P}_- \times \mathcal{P}_+ \times \mathbb{R} : (U_-, U_+, n) \mapsto (U_-, U_+, s),
$$

where the mapping $(U_-, U_+, n) \mapsto (U_-, U_+, s)$ is defined by the following steps:

1. Map the states $U_\pm$ in direction of $n$, according to (9), resulting in $u_\pm = U_{\pm,n,\pm}$ and $U_{\pm,n,\pm}$.
2. Evaluate the surrogate solver (for $R$)

$$(u_{\pm,n,-}, u_{\pm,n,+}, s) = R_\partial(u_-, u_+).$$

3. Project the directional states:

$$U_{\pm}^* = P_n(u_+^*) + U_{\pm,n,\pm},$$

with $P_n$ defined as in (10).
- Run the FV-IPMM-method (Algorithm 4) with the initial mesh $\mathcal{T}(0)$, the initial data $(U_i^0)_{i \in I_{\pm}(0)}$, and using the interface solver $\mathcal{R}$ defined in (27).

**Result:** approximate solution $U_{\mathcal{T}}(x, t) = U_i^n$, for $x \in C_i(t_n)$ and $t \in [t_n, t_{n+1})$, on the moving mesh $\mathcal{T}(t)$.

**Remark 9 (Active Sampling Strategies).** One major drawback of a fixed surrogate solver is, that it is restricted to a predetermined flow regime, which is due to the static bounding region $B_{\mathcal{R}}$.

To mitigate this problem, it is possible to employ an active sampling strategy, which in this...
context is presented in our work [38]. The main idea is to introduce a score value for an arbitrary input \( \mathbf{x} \), that measures the uncertainty of the surrogate solver on this specific input point. If the uncertainty becomes too large, the original solver is evaluated, and the resulting sample \( \mathbf{y} = \mathcal{R}(\mathbf{x}) \) is added to the training data set and the surrogate solver gets retrained.

### 6.2. The Multiscale Model for Liquid–Vapor Flow

In this section, we apply the multiscale model to compressible liquid–vapor flow. On the continuum scale, the two-phase flow is described by the Euler equations, as in Section 2.2. The interface dynamics on the microscale are modeled by MD simulations, see Section 3.2, which are substituted by a surrogate solver, see Section 4.

**Microscale Interface Solver** To define the liquid–vapor flow multiscale model, we have to specify a microscale interface solver of the form

\[
R : (\rho_\ast, m_\ast, E_\ast, \rho_\ast', m_\ast', E_\ast') \mapsto (\rho_\ast, m_\ast, E_\ast, \rho_\ast', m_\ast', E_\ast').
\]  

The solver (28) is based on MD simulations (Section 3.2) and is realized as follows. Let the initial states \((\rho_\ast, m_\ast, E_\ast) \in \mathcal{R}_\ast, (\rho_\ast, m_\ast, E_\ast) \in \mathcal{R}_\ast\), and normal direction \( \mathbf{n} \in \mathbb{S}^{d-1} \) be given. We compute the specific inner energies \( \varepsilon_\ast = \frac{E_\ast}{\rho_\ast} + \frac{|m_\ast|^2}{2\rho_\ast} \), and temperatures \( T_\ast = T(\rho_\ast, \varepsilon_\ast) \), using the EOS. Next, map \((\rho_\ast, m_\ast, T_\ast)\) in direction of \( \mathbf{n} \) according to (9), i.e.

\[
\begin{aligned}
\mathbf{u}_\ast &= (\rho_\ast, m_\ast, T_\ast) := U_{|\mathbf{n} = \mathbf{n}} = (\rho_\ast, m_\ast \cdot \mathbf{n}, T_\ast), \\
U_{|\mathbf{n} = \mathbf{n}} &= (0, m_\ast - (m_\ast \cdot \mathbf{n}) \mathbf{n}, 0).
\end{aligned}
\]

Then, we consider the fluid states in the reference frame of Section 4.1, which means we choose the --phase velocity \( \mathbf{v}_\ast = \frac{m_\ast}{\rho_\ast} \) as the reference velocity \( \mathbf{v}_{\text{ref}} := \mathbf{v}_\ast \). Thus, we consider the fluid velocities \( \mathbf{v}_\ast = \mathbf{v}_\ast - \mathbf{v}_{\text{ref}} \). Next, we solve the Riemann problem (11) in normal direction \( \mathbf{n} \) with Riemann data \((\rho_\ast, \mathbf{v}_\ast, T_\ast)\) using the microscale model. This means, we run a MD simulation (Algorithm 3) for the initial continuum states \((\rho_\ast, \mathbf{v}_\ast, T_\ast)\) and \((\rho_\ast, \mathbf{v}_\ast, T_\ast)\), i.e. evaluating \( R_{\text{MD}} \) from (19) resulting in the states \((\rho_\ast', m_\ast', T_\ast')\), \((\rho_\ast', m_\ast', T_\ast')\), and the wave speed \( \tilde{s} \) in normal direction \( \mathbf{n} \), with \( m_\ast' \), \( m_\ast' \), \( \rho_\ast' \). Next, we return the reference velocity, i.e.

\[
m_\ast' = m_\ast' + \rho_\ast' \mathbf{v}_{\text{ref}}, \quad \tilde{s} = \tilde{s} + \mathbf{v}_{\text{ref}}.
\]

The directional states are projected back to the full-dimensional states

\[
(\rho_\ast', m_\ast', T_\ast') = (\rho_\ast', m_\ast' \mathbf{n}, T_\ast') + U_{|\mathbf{n} = \mathbf{n}} = P_n((\rho_\ast', m_\ast', T_\ast')) + U_{|\mathbf{n} = \mathbf{n}}.
\]

with \( P_n \) defined as in (10). After computing the specific inner energies \( \varepsilon_\ast' = \varepsilon(\rho_\ast', T_\ast') \), and energy densities \( E_\ast' = \rho_\ast' \varepsilon_\ast' + \frac{|m_\ast'|^2}{2\rho_\ast'} \), we have obtained the wave states \((\rho_\ast', m_\ast', E_\ast')\), \((\rho_\ast', m_\ast', E_\ast')\), and the interface speed \( s \) in direction of \( \mathbf{n} \).

### 7. Numerical Simulation Results for the Liquid–Vapor Multiscale Model

In this section, we present simulation results for the liquid–vapor multiscale model, as presented in Section 6.2. We consider the two-phase Euler equations of Section 2.2 on the continuum scale, and MD simulations for the phase boundary dynamics, as described in Section 3.2. The parameters used for the simulations are found in Appendix D.
A Molecular–Continuum Multiscale Model for Inviscid Liquid–Vapor Flow with Sharp Interfaces

Figure 9: Multiscale simulation for the liquid–vapor flow model in one space dimension, overlaid over the corresponding MD simulations (averaged over 50 simulations and 200 bins for local averaging, the shaded region indicates the standard error of the mean). The initial conditions \( \rho_\text{L} = 0.65, T_\text{L} = 1.0, \rho_\text{V} = 0.05, T_\text{V} = 1.0, v_\text{L} = 0 \) yield a pressure-driven wave that expands from the liquid phase.

7.1. One-dimensional Results

To start with, we consider the multiscale model in one space dimension. Our focus lies hereby on Riemann problems, which are compared with their corresponding MD simulation result. In that way, we are able to validate the consistency of the multiscale model with respect to the microscale MD simulations. We consider two settings: a pressure-driven wave and a vapor wave colliding with liquid fluid.

**Pressure-Driven Wave** For the first example, we consider the Riemann initial data \( \rho_\text{L} = 0.65, T_\text{L} = 1.0, \rho_\text{V} = 0.05, T_\text{V} = 1.0, v_\text{L} = 0 \). As seen in Figure 9, this results in a pressure-driven wave, expanding the liquid phase. Due to the expansion and the decreasing density inside the liquid phase, the temperature decreases as well. The vapor next to the interface is slightly compressed and therefore has a slightly increased temperature. Some mass is transferred from the vapor into the liquid phase, according to the (averaged) relative mass flux \( j \approx -0.004 \).

The MD simulation results are close to the multiscale solution. Specifically, this holds for the speed of the observed waves. The wave plateau values for the density coincide very well — however, for the velocity and the temperature, there is some discrepancy. This is due to the higher fluctuation in statistical averaging on the vapor side, as on the molecular-scale fewer particles are present there.
Figure 10.: Multiscale simulation for the liquid–vapor flow model in one space dimension and the corresponding MD simulations (averaged over 50 simulations and 200 bins for local averaging, the shaded region indicates the standard error of the mean). The initial conditions $\rho_\text{L} = 0.58$, $v_\text{L} = 0$, $T_\text{L} = 1.0$, $\rho_\text{v} = 0.05$, $v_\text{v} = -0.5$, $T_\text{v} = 1.0$ are chosen in such a way that a vapor phase wave hits the liquid phase.

**Liquid–Vapor Collision** In the second example, we consider a vapor wave that hits the liquid phase. This is implemented by the Riemann initial data $\rho_\text{L} = 0.58$, $v_\text{L} = 0$, $T_\text{L} = 1.0$, $\rho_\text{v} = 0.05$, $v_\text{v} = -0.5$, $T_\text{v} = 1.0$. The simulation result is shown in Figure 10. In the vapor phase we observe an increase in the density and temperature. The averaged relative mass flux $j \approx -0.006$ indicates that mass is transferred into the liquid phase. In the same way, the momentum of the vapor wave is transferred into the liquid phase as seen in the velocity plot. Most noteworthy is the excellent correspondence between multiscale solution and MD simulation — only the temperature distribution of the MD simulation exhibits some heat conductivity effects which are not present on the continuum scale.

In summary, the one-dimensional simulation results are consistent with the molecular-scale simulations, which validates the multiscale approach.

### 7.2. Two-dimensional Results

Next, we show the performance of the multiscale model in two space dimensions. For this scenario we consider the domain $\Omega = (-1.5, 1.5)^2$, with the two initial phase domains $\Omega_\text{L}(0) = \{ x = \ldots$
\((x_1, x_2) \in \mathbb{R}^2 : |x_2| < 0.15\) and \(\Omega_+(0) = \Omega \setminus \overline{\Omega}_-(0)\). The initial data in each subdomain is set to

\[
(\rho, \mathbf{v}, T)(x, t) = \begin{cases} 
(0.62, (0, 0), 0.8), & x \in \Omega_-(0), \\
(0.06, (0, 0), 0.9), & x \in \Omega_+(0) \text{ and } x_1 \geq -0.5, \\
(0.06, (0.5, 0), 0.9), & x \in \Omega_+(0) \text{ and } x_1 < -0.5.
\end{cases}
\]

This corresponds to a vapor wave colliding with a liquid droplet. At the left boundary, at \(x_1 = -1.5\), we apply inflow boundary conditions by setting ghost cell values to \((\rho, \mathbf{v}, T) = (0.06, (0.5, 0), 0.9)\). Everywhere else we use outflow boundary conditions by copying the inner cell values to the ghost cells.

The density and velocity field of the simulation are shown in Figure 11 for several time steps; the corresponding temperature field is depicted in Figure 12. As the initial data does not coincide with equilibrium states, the liquid droplet starts to oscillate — clearly seen in Figure 13. Meanwhile, the vapor wave increases the density and temperature inside the vapor phase. Then, the vapor wave interacts with the droplet, deforms it, and transports it through the domain. Note that surface tension is not included in the model. Therefore, the droplet deformation might be exaggerated.

### 7.3. Three-dimensional Results

In this section we demonstrate the capabilities of the multiscale model in three space dimensions. For this setting we consider the domain \(\Omega = (-1.5, 1.5)^3\), with the two initial phase domains \(\Omega_-(0) = \{x = (x_1, x_2, x_3) \in \mathbb{R}^3 : |x_2| < 0.15\} \text{ and } \Omega_+(0) = \Omega \setminus \overline{\Omega}_-(0)\). For the initial data in each subdomain we set

\[
(\rho, \mathbf{v}, T)(x, t) = \begin{cases} 
(0.62, (0, 0, 0), 0.8), & x \in \Omega_-(0), \\
(0.06, (0, 0, 0), 0.9), & x \in \Omega_+(0) \text{ and } x_1 \geq -0.5, \\
(0.06, (0.5, 0, 0), 0.9), & x \in \Omega_+(0) \text{ and } x_1 < -0.5.
\end{cases}
\]

For the boundary conditions we apply inflow boundary conditions at \(x_1 = -0.5\) with the ghost cell value \((\rho, \mathbf{v}, T) = (0.06, (0.5, 0, 0), 0.9)\). Everywhere else outflow boundary conditions are applied. This corresponds to the two-dimensional setting of Section 7.2.

The simulation results are presented in Figure 14 for the density and the velocity, and in Figure 15 for the temperature. Both figures show two-dimensional slices through the three-dimensional domain at \(x_3 = 0\).

Initially, the liquid droplet is not in equilibrium with the surrounding vapor, resulting in a compression of the droplet. Then the vapor wave hits the droplet and acts upon it, moving the droplet through the computational domain. During that phase the droplet slowly grows in volume. Compared to the two-dimensional simulation, the deformation is much less pronounced. The reason for this is that the fluid has more freedom to move around the droplet, as opposed to the two-dimensional setting.

Finally, let us regard the computational effort of the simulation.\(^1\) Averaged over the \(1 \cdot 10^4\) time steps the mesh consists of \(2.7 \cdot 10^8\) cells with \(3.6 \cdot 10^5\) interface facets. Each time step takes approximately 24 s, split into 2 s for performing the moving mesh operations, and 22 s for computing the finite volume fluxes and updates. In total (including I/O), this amounts to a runtime of about

---

\(^1\) All computations in this contribution are performed on a desktop computer equipped with an AMD Ryzen Threadripper 2950X 16-core processor and 128 GB RAM.
Figure 11: Two-dimensional multiscale simulation of the liquid–vapor flow model. The subfigures depict the density and velocity of the solution at various time steps. The corresponding plots of the temperature can be found in Figure 12.
Figure 12.: Two-dimensional multiscale simulation of the liquid–vapor flow model. The subfigures depict the temperature of the solution at various time steps. The corresponding plots of the density can be found in Figure 12.
Figure 13.: Time evolution of the volume of the droplet and the droplet-averaged density for the liquid–vapor flow multiscale simulation in two space dimensions.

66 h. We note however, that the implementation is mostly serial. Only the numerical fluxes in the bulk phases are computed in parallel with a shared memory parallelization. Employing the surrogate solver is crucial to run this simulation. If we would not have used a surrogate solver, we would need to perform a MD simulation at every interface facet. This would sum up to 600 h of (albeit parallelizable) computation time for every time step. Compared to that, a single evaluation of the surrogate solver takes only 0.2 ms. Even including the offline phase for generating the surrogate solver (approximately 3000 h), the additional effort is quickly redeemed after a few time steps.

8. Conclusion

Up to now, there were few methods to solve inviscid, temperature-dependent liquid–vapor flow, and the applied methods tend to be either computationally expensive, or are unable to render some physical effects accurately. We developed a multiscale model for inviscid liquid–vapor flow, that accounts for the interface physics on the molecular scale. For this purpose, we utilize the molecular-scale Riemann problem as the link between the continuum scale and the molecular scale. Thereby, we avoid explicit closures on the continuum scale which are in many cases not valid nor available. This enables us to simulate fluid regimes that could not be considered before. However, that comes at the cost of expensive MD simulations. We overcome this bottleneck by employing (mass-conservative) constraint-aware neural networks, rendering the whole implementation efficient enough to perform simulations in two and three space dimensions.

Notwithstanding, there remain some open questions.

• An important addition to the continuum-scale model and its discretization is to include topological changes such as droplet merging or splitting. For this purpose, sub-models have to be introduced that describe these phenomena in a sharp-interface setting. In the same way, nucleation and vanishing droplets should be supported.

• Surface tension effects are an important factor in two-phase flows. However, they are not taken into account in this work. It is straightforward to include them on the continuum scale because the curvature of the phase boundary can be computed directly from the geometric
Figure 14: Slice through the three-dimensional multiscale simulation of the liquid–vapor flow model at $x_3 = 0$. The sub-figures depict the density and velocity of the solution at various time steps. The corresponding plots of the temperature can be found in Figure 15.
Figure 15.: Slice through the three-dimensional multiscale simulation of the liquid–vapor flow model at $x_3 = 0$. The sub-figures depict the temperature of the solution at various time steps. The corresponding plots of the density can be found in Figure 15.
information of the sharp interface in the mesh, see e.g. [47]. The local curvature can then be passed to the microscale interface solver. We have already demonstrated this in our work [10]. What is missing is an efficient method to include the curvature effects on the molecular scale.

• We consider inviscid flow without heat conduction in the bulk phases. To include heat and viscous fluxes it suffices to exchange the Euler equations by the Navier–Stokes–Fourier system, while inferring the derivatives of the fluid states from the MD simulations.

Regarding the surrogate solver, various improvements can be made. Active learning may be utilized and more efficient sampling strategies for generating the training data set can be applied. The timescale of the MD simulations and the size of the interface sampling region are determined heuristically. This is due to the lack of rigorous theory for bridging the scales from the molecular scale particle model to a continuum-scale fluid model.

The multiscale model can be naturally extended to the case of more complex fluids, only the MD simulations have to be adapted for the corresponding molecule types. Moreover, the multiscale model is easily adaptable to various sharp-interface, two-phase flow scenarios, while preserving the physical accuracy of MD simulations.

Acknowledgements

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) through the project SFB–TRR 75 “Droplet dynamics under extreme ambient conditions” with the project number 84292822 and the DFG under Germany’s Excellence Strategy - EXC 2075 with the project number 390740016.

Appendix A. Elements of Molecular Dynamics Simulations

In this section, we explain the tailored MD simulations in detail. Starting with the cutoff-potential substituting the full Lennard-Jones potential (16). Subsequently, we give a detailed explanation of other MD elements, namely the particular choice of the computational domain, the long-range corrections, and the thermostat.

Cutoff-Potential Solving the equation of motion (17) directly requires the computation of \((N_p - 1)^2\) pair-interactions. For a high number of particles this becomes unfeasible. To respond to this problem, it is common to apply a localized potential by introducing a cutoff length \(r_{\text{cutoff}} > 0\), above which the potential is set to zero, i.e.

\[
\phi(r; r_{\text{cutoff}}) := \begin{cases} 
\phi(r), & r < r_{\text{cutoff}}, \\
0, & r \geq r_{\text{cutoff}}.
\end{cases}
\]  

(30)

Naturally, using a cutoff-potential comes at a cost: long-range forces are not considered anymore. They have to be re-included via long-range corrections or long-range forces, which will be discussed later.
Computational Domain  We are interested in simulating the fluid dynamics in normal direction of the interface. Therefore, it is beneficial to consider an adapted computational domain, that is large in $x$-direction and small in the transversal directions, see Figure 16 b. In the transversal directions we consider periodic boundary conditions, whereas in $x$-direction reflecting boundary conditions are implemented. We fix the height $h_{\text{dom}}$ in the transversal directions to a multiple of the cutoff-radius $r_{\text{cutoff}}$ and set the length $l_{\text{dom}}$ in $x$-direction according to the desired fluid density.

![Diagram of computational domains]

(a) Standard, cubic computational domain  (b) Elongated cuboid domain with linked-cell approach

Figure 16.: Two different setups for the computational domain.

Linked-Cell Approach  The big advantage of choosing the adapted computational domain is that, in conjunction with the cutoff-potential, we are now able to consider one-dimensional grid cells with cell length $\Delta c > r_{\text{cutoff}}$ for a more efficient and faster computation of the interaction forces. This is achieved by computing only the interactions of all particles within a cell and its two neighboring cells. In that way, not all pairwise particle distances have to be computed, which saves computational resources. This approach is called linked-cell or linked-list approach. Due to the one-dimensional nature of the grid, we can distribute the particles onto the cells by sorting them according to their $x$-coordinate, which is done every $\chi_{\text{sort}} \in \mathbb{N}$ time steps.

Long-Range Forces  Following the approach described in [27], we compute the forces depending on the cell-wise local density average. Therefore, the cell-averaged densities $\bar{\rho}_{C_k}$ are computed at each point in time for every cell $C_k$. Then, a particle $i$ in cell $C_k$ is subject to the long-range force

$$f_{C_k,LR} = \sum_{l=k-1,k,k+1} 2\pi (c_k - c_l) \phi(|c_k - c_l|) \bar{\rho}_{C_l} \Delta c \quad \text{for } x_i \in C_k,$$

where $c_l$ denotes the center point of cell $C_l$. Note that the long-range forces do not depend on the particle position and can therefore be precomputed for each cell.

Statistical Averaging  By solving the MD system (17), microscale information about the fluid system is generated. In most situations, the trajectory of every single particle is not of interest. Instead, the focus lies on the continuum-scale behavior of the fluid. To convert the microscale dynamics to continuum-scale quantities, we apply methods from the domain of statistical mechanics. By employing the Irving–Kirkwood formulas [26] it is possible to formulate microscopic distributions of some key quantities. Thus, the microscopic instantaneous density $\hat{\rho}(x, \tau)$ and momentum $\hat{\mathbf{m}}(x, \tau)$ distributions are given by

$$\hat{\rho}(x, \tau) = \sum_{i=1}^{N_\tau} m_i \delta_{\chi(\tau)}(x), \quad \hat{\mathbf{m}}(x, \tau) = \sum_{i=1}^{N_\tau} m_i \mathbf{v}_i(\tau) \delta_{\chi(\tau)}(x),$$
where \( m_i > 0, x_i \in \mathbb{R}, \mathbf{u}_i \in \mathbb{R}^3 \) are the mass, position in \( x \)-direction and velocity of the \( i \)-th particle. Here, \( \delta_{x_i(\tau)}(\cdot) \) denotes the Dirac delta distribution centered at \( x_i \). The local temperature distribution can be formally understood as

\[
\tilde{T}(x, \tau) := \frac{1}{d} \left( \sum_{i=1}^{N_p} \delta_{x_i(\tau)}(x) \right)^{-1} \sum_{i=1}^{N_p} (m_i (\mathbf{v}_i - \mathbf{\bar{v}}) \cdot (\mathbf{v}_i - \mathbf{\bar{v}})) \delta_{x_i(\tau)}(x).
\]  

(32)

However, the temperature is only defined as an averaged quantity and is undefined in the absence of particles.

**Thermostat** To provide meaningful MD simulations, it is important to control the temperature. Typically, this is done via a so-called thermostat during the simulations. We will apply one of the most basic thermostats: the velocity rescaling thermostat, see e.g. [9]. It controls the temperature by rescaling the relative velocity \((\mathbf{v}_i - \mathbf{\bar{v}})\) of every particle by

\[
\lambda = \sqrt{\frac{\tilde{T}_{\text{ref}}}{\tilde{T}_{\text{inst}}}},
\]

where \( \tilde{T}_{\text{ref}} \geq 0 \) is the prescribed target temperature and \( \tilde{T}_{\text{inst}} \) is the measured, instantaneous temperature, averaged over the whole system using (32).

**MD Time-Step Algorithm** The following algorithm describes the computation of a single time step of the MD simulation. We use the Velocity Verlet algorithm [57] to solve (17) for a time step from \( \tau_n = n \Delta \tau \) to \( \tau_{n+1} \), with \( \Delta \tau > 0 \) and \( n \in \mathbb{N} \).

---

**Algorithm 10: Molecular Simulation Time-Step with Linked Cells**

- **Input:** particle state arrays \( \mathbf{x}[i] = x_i(\tau_n), \mathbf{v}[i] = \mathbf{v}_i(\tau_n), \) and \( \mathbf{a}[i] = a_i(\tau_n) \) for \( i = 1, \ldots, N_p \) at \( \tau_n \geq 0 \).

- **Parameters:** time step \( \Delta \tau > 0 \), sorting frequency \( \chi_{\text{sort}} \in \mathbb{N} \), linked-cell size \( \Delta c > 0 \).

- **Algorithm**

  - If \( n \mod \chi_{\text{sort}} = 0 \): sort particles w.r.t. \( x \)-coordinates and store particles indices for each cell \( C_k \).
  
  - Compute cell-averaged densities \( \mathbf{\bar{p}}_{C_k} \) and the long-range forces \( \mathbf{f}_{C_k,\text{LR}} \) for each cell \( C_k \), see (31).
  
  - Update positions, for all \( i = 1, \ldots, N_p \),
    
    \[
    \mathbf{x}[i] += \Delta \tau \mathbf{v}[i] + \frac{1}{2} \Delta \tau^2 \mathbf{a}[i].
    \]

  - Apply boundary conditions.

  - Compute intermediate velocities, for all \( i = 1, \ldots, N_p \)
    
    \[
    \mathbf{v}[i] += \frac{1}{2} \Delta \tau \mathbf{a}[i].
    \]

  - Update accelerations, for all \( i = 1, \ldots, N_p \),
    
    \[
    \mathbf{a}[i] = \frac{1}{m_i} (\mathbf{f}_i + \mathbf{f}_{C_k,\text{LR}}).
    \]
using the pair-interaction potential \( \Phi \), the (updated) positions \( x[i] \), and equations (30), (31).

• Update velocities, for all \( i = 1, \ldots, N_p \),

\[
\mathbf{v}[i] + = \frac{1}{2} \Delta t \mathbf{a}[i].
\]

Result: \( x(i, \tau_{n+1}) = x[i], \mathbf{v}(i, \tau_{n+1}) = \mathbf{v}[i], \) and \( \mathbf{a}(i, \tau_{n+1}) = \mathbf{a}[i] \).

**Molecular-Scale Riemann Problem**  The following algorithm describes the initialization of the molecular-scale Riemann problem in detail.

### Algorithm 1: Molecular-Scale Riemann Problem Initialization

**Input:** continuum fluid states \((\rho_-, \mathbf{v}_-, T_-)\), \((\rho_+, \mathbf{v}_+, T_+)\).

**Parameters:** total number of particles \( N_p \), number of thermalization steps \( N_{\text{therm.steps}} \), thermostat frequency \( \chi_{\text{th}} \), domain width factor \( w \), maximum particle ratio \( \alpha_{\text{max}} \), domain length ratio \( \alpha_{\text{dom}} \).

**Algorithm**

- Particle ratio: \( \alpha = (1 + \alpha_{\text{dom}} \rho_+ / \rho_-)^{-1} \), clamped at \( 1 - \alpha_{\text{max}} \) and \( \alpha_{\text{max}} \).
- Number of particles in \( \pm \)-phase: \( N_\pm = \alpha N_p \).
- Domain size in transversal directions: \( l_{yz} = w \sigma \sigma_{\text{cutoff}} \).
- Volume of each phase-subdomain \( \Omega_\pm \): \( \text{vol}_\pm = \frac{m_0 N_\pm}{\rho_\pm} \).
- Domain width for each phase subdomain \( \Omega_\pm \) in \( x \)-direction: \( l_x = \frac{\text{vol}_\pm}{l_{yz}} \).
- Interface gap to avoid overlapping particles: \( l_{\text{gap}} = 2^{\frac{3}{2}} \sigma \).
- Define the domains

\[
\begin{align*}
\Omega_\pm & := [0, L] \times [0, l_{yz}]^2, & \Omega_{-\text{gap}} & := [0, L - \frac{1}{2} l_{\text{gap}}] \times [0, l_{yz}]^2, \\
\Omega_\pm & := [L, L + L] \times [0, l_{yz}]^2, & \Omega_{+\text{gap}} & := [L + \frac{1}{2} l_{\text{gap}}, L + L] \times [0, l_{yz}]^2.
\end{align*}
\]

- Distribute \( N_\pm \) particles on a uniform lattice in \( \Omega_{\pm, \text{gap}} \). If particles remain, distribute them randomly, while avoiding overlaps (using Algorithm 14).
- Thermalization phase: run \( N_{\text{therm.steps}} \) of a MD simulation in each subdomain \( \Omega_\pm \), and each \( \chi_{\text{th}} \)-th step apply a thermostat with target temperature \( T_\pm \).
- Remove average velocities and add continuum fluid velocities \( \mathbf{v}_\pm \) to all particles in \( \Omega_\pm \).
- Combine domains \( \Omega := \Omega_- \cup \Omega_+ \).

**Result:** \( N_p \) particles in a microscale Riemann problem configuration with corresponding continuum fluid states \((\rho_-, \mathbf{v}_-, T_-)\) and \((\rho_+, \mathbf{v}_+, T_+)\).
Molecular-Scale Interface Tracking  The following algorithm describes the tracking of the phase boundary on the molecular scale.

**Algorithm 12: Molecular-Scale Interface Tracking**

**Input:** particle positions $x_i(\tau_n)$, $i = 1, \ldots, N_p$, at time $\tau_n > 0$, $n \in \mathbb{N}$; former interface positions $\Gamma(t_j)$ for $j = \max(n - n_{buffer}, 0), \ldots, \max(n - 1, 0)$.

**Parameters:** RBF kernel parameter $\gamma_{RBF}$, interface tracking width ratio $\alpha_{track}$, buffer size $n_{buffer}$.

**Algorithm**

- Get the new interface position $\Gamma(\tau_n)$ by solving the optimization problem in $B_{\omega_{\text{track}}}(\Gamma(\tau_{n-1})) := (\Gamma(\tau_{n-1}) - \omega_{\text{track}}, \Gamma(\tau_{n-1}) + \omega_{\text{track}}) \subset \mathbb{R}$, with $\omega_{\text{track}} := \alpha_{\text{track}} \lambda_{\text{dom}}$,

  $$\Gamma(\tau_n) = \arg \max_{x \in B_{\text{offset}}(\Gamma(\tau_{n-1}))} |\partial_x \hat{\rho}(x, \tau_n)|.$$

- If $n \geq 2$, perform linear regression over the data $(\tau_j, \Gamma(\tau_j)), j = \max(n - n_{buffer}, 0), \ldots, n$.

The slope $s$ of the fitted line is the averaged interface speed $s(\tau_n) = s$.

**Result:** interface position $\Gamma(\tau_n)$ and interface speed $s(\tau_n)$.

Appendix B. Sampling Algorithms

**Uniform Sampling in a Convex Domain**  We seek to draw random samples inside a convex domain $\Omega \subset \mathbb{R}^d$. Commonly, this is done via rejection sampling, but this can become very expensive in high-dimensional spaces. Alternatively, we perform uniform sampling on a mesh of the convex domain. Here we exploit the fact that the Dirichlet-distribution yields a uniform point distribution after projection on a simplex.
Algorithm 13: Convex Set Sampling

**Input:** point cloud $P = \{p_0, \ldots, p_M\} \subset \mathbb{R}^d$, number of new samples $N$.

**Algorithm**

- Construct convex hull of $P$.
- Construct $d$-dimensional Delaunay mesh of the points on the convex hull.
- Calculate the volume of each simplex in the mesh.
- For $i = 1, \ldots, N$ do
  - $T_i \leftarrow$ choose a random simplex from the Delaunay mesh with probability proportional to the simplex volume.
  - $\alpha \leftarrow$ Get a $(d + 1)$-dimensional vector drawn from a uniform Dirichlet distribution.
  - $x_i \leftarrow$ linear combination of the vertices of $T_i$ with the coefficients of $\alpha$.

**Result:** $\{x_1, \ldots, x_N\}$.

*Distance-Maximizing Sampling* Uniform random sampling might result in bad space-filling behavior, especially for sparse data in high-dimensional spaces. We use distance-maximizing sampling using a multidimensional variant of the best-candidate algorithm of [42]. However, by using the standard euclidean metric to measure the distance this can lead to a high concentration of samples near the domain boundary [48]. Instead, we employ the Minkowski distance

$$\text{dist}(x, y) := \left( \sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

with $p \in (0, 1)$. This choice appears to provide a much better performance in high dimensional spaces [3] and avoids a high concentration of the points near the domain boundary. As a rule of thumb, we set $p = 4^{-d}$. This empirical choice for $p$ appears to produce the desired results.
Appendix C. Surrogate Solver Details for the Liquid–Vapor Flow Model

In this section, we provide some details regarding the surrogate solver for liquid–vapor flow discussed in Section 6.2, namely the data set generation and the outline of the training procedure.

Data Set Generation To be able to train a surrogate solver, an appropriate data set has to be generated. The general outline of the procedure is described in Section 4. First, we have to define the model-specific data range for the input data set $D_{in}$. Each input data point is given by a tuple $(\rho_-, \nu_-, T_-, \rho_+, \nu_+, T_+)$, with $\nu_- = 0$ after transforming the reference frame. The range of the $-$-phase densities $\rho_-$ and temperatures $T_-$ is defined by the convex set formed by the following points

\[(\rho_{\text{min},-}, T_{\text{min}}), (\rho_{\text{min},+}, T_{\text{max}}), (\rho_{\text{max},+}, T_{\text{min}}), (\rho_c, T_c), (\rho_c, T_{\text{max}}),\]

with $\rho_{\text{min},+} = 10^{-4}$, $\rho_{\text{max},+} = 0.02$, $\rho_c = 0.31$, $T_{\text{min}} = 0.4$, $T_c = 1.32$, and $T_{\text{max}} = 2.5$. Analogously, the $+$-phase range for the densities $\rho_+$ and temperatures $T_+$ is given by the convex set formed by the points

\[(\rho_{\text{min},+}, T_{\text{min}}), (\rho_c, T_c), (\rho_{\text{max},+}, T_{\text{min}}), (\rho_{\text{max},-}, T_c), (\rho_c, T_{\text{max}}),\]

with $\rho_{\text{min},-} = 0.7$, and $\rho_{\text{max},-} = 1.0$. Finally, the $+$-velocity lies in the interval $\nu_+ \in (\nu_{\text{min}}, \nu_{\text{max}})$, with $\nu_{\text{min}} = -2.5$, and $\nu_{\text{max}} = 1.5$. Altogether, this gives the definition of the input bounding domain $B_{in}$.

Inside the input bounding domain, $N_{in} = 6000$ data points are sampled, using Algorithm 14, while exploiting that $\nu_- = 0$, i.e. the sampling domain is five-dimensional. This results in the initial data set $D_{in} = \{(\rho_-, \nu_-, 0, T_, \rho_+, \nu_+, T_+): i = 1, \ldots, N_{in}\}$.

For each $(\rho_-, \nu_-, 0, T_, \rho_+, \nu_+, T_+) \in D_{in}$ we evaluate the MD interface solver (Algorithm 3), i.e.

\[R_{MD}(\rho_-, \nu_-, 0, T_, \rho_+, \nu_+, T_+) = (\rho_-^*, m_-^*, T_-^*, \rho_+^*, m_+^*, T_+^*, s).\]  

(33)
We gather the valid output in the output data set $D_{\text{out}} = \{(\rho^\ast, m^\ast, T^\ast, \rho^\prime, m^\prime, T^\prime, s)_i : i = 1, \ldots, N_{\text{data}}\}$ and identify each output sample with its corresponding input $(\rho, v_\ast = 0, T, \rho^\prime, v^\prime, T^\prime)$, resulting in the complete data set

$$D = \left\{ \left( (\rho, v_\ast = 0, T, \rho^\prime, v^\prime, T^\prime) \right)_i : i = 1, \ldots, N_{\text{data}} \right\}.$$ 

**Remark 15 (Repeating Molecular Dynamics Simulations).** As the MD simulations are non-deterministic and subject to noise, we repeat the MD simulation, i.e. the evaluation of (33), for each data point 3 times. This results in a larger data set and gives us a rough indicator of the accuracy of the MD simulations. Which — implicitly — can help us to avoid overfitting during the neural network training.

The data set is available at [36]. To get an overview of the data set, the resulting data set is illustrated in Figure 17, where we plot the output values over the input values, including the range of the output values, which is indicated by error bars.

A single run of a single MD simulation (i.e. Algorithm 3) takes between 9 and 11 min. For the whole data set this amounts to roughly 3000 h of computing time. The sampling of the data points is however fully parallelizable, and therefore, the workload can be easily split among numerous machines.

**Neural Network Training** The training of the neural network proceeds in the usual way and follows our approach used in [37] for CRes-networks. For training and employing we use the neural network framework PyTorch [45]. The particular parameters for this model are outlined in the following. We use the data set $D$ of $C$ and split it into a training data set $D_{\text{train}}$ with 14 889 samples and a validation data set $D_{\text{val}}$ with 2976 samples. We ensure that all repetitions of the MD simulations (see Remark 15) belong to either the training or the validation data set. The neural network consists of 5 hidden layers with 50 nodes each. The learning rate of the Adam optimizer [29] is set to $10^{-3}$, the weight decay parameter $\alpha_{\text{w}}$ to $10^{-3}$, and the gradient norm clipping parameter to $10^3$. The training and validation errors during the training procedure are shown in Figure 18.
A Molecular–Continuum Multiscale Model for Inviscid Liquid–Vapor Flow with Sharp Interfaces

Figure 17.: The data set for the liquid–vapor flow multiscale model. The data set consists of 17,865 data points (with 5,955 unique initial states). The scatter plots show the relation of the input states $\rho_-, T_-, \rho_+, T_+$ to the wave states $\rho'^-, v'^-, \rho'^+, v'^+, T'^+, T'_-$, and wave speed $s$. The error-bars show the data range of each separate MD simulation. The input variable $v$ is not shown as it equals zero throughout the data set.

Figure 18.: Evolution of the loss on the training and the validation data set during the training of the neural network surrogate solver for the liquid–vapor flow multiscale model.
Appendix D. Temperature-Dependent Multiscale Model: Parameter Tables

| Molecular Dynamics |
|-------------------|
| **Lennard–Jones potential** |
| \( \sigma \) | 1 |
| \( \varepsilon \) | 1 |
| cutoff-radius \( r_{\text{cutoff}} \) | 2.5 |
| particle mass \( m_0 \) | 1 |

| Simulation |
|-----------|
| number of particles \( N_p \) | 32,768 |
| number of time steps \( N_{\text{end}} \) | \( 10^5 \) |
| processing frequency \( \chi_{\text{pr}} \) | 100 |
| time step \( \Delta t \) | \( 10^{-3} \) |
| domain width factor \( w \) | 3.5 |
| domain length ratio \( \alpha_{\text{dom}} \) | 3.0 |
| maximum particle ratio \( \alpha_{\text{max}} \) | 0.99 |
| linked-cell size \( \Delta c \) | 1.2 \( \sigma r_{\text{cutoff}} \) |
| sorting frequency \( \chi_{\text{sort}} \) | 20 |
| thermalization steps \( N_{\text{therm.steps}} \) | \( 5 \cdot 10^2 \) |
| thermalization frequency \( \chi_{\text{th}} \) | 100 |
| particles per thermostat cell \( n_{\text{therm.L}} \) | 500 |
| particles per thermostat cell \( n_{\text{therm.R}} \) | 50 |
| time sampling ratio \( \alpha_{\text{t-smpl}} \) | 0.2 |
| sampling width \( w_{\text{sr}} \) | 50 |
| sampling interface offset \( o_{\text{sr}} \) | 2.5 |
| interface tracking buffer size \( n_{\text{buffer}} \) | 5 |
| interface tracking width \( \alpha_{\text{track}} \) | 0.1 |
| RBF parameter \( y_{\text{RBF}} \) | \( 5 \cdot 10^{-4} \) |

Table 1.: Parameter table for the molecular dynamics simulation in case of the temperature-dependent, two-phase flow multiscale model.

| FV-IPMM 1d |
|------------|
| end time \( t_{\text{end}} \) | 0.1 |
| time step \( \Delta t \) | \( 10^{-5} \) |
| domain \( \Omega \) | \((-1, 1)\) |
| base cell width \( \Delta x \) | \( 10^{-3} \) |
| \( \Delta x_{\text{min}} \) | \( \frac{1}{2} \Delta x \) |
| \( \Delta x_{\text{max}} \) | \( \frac{3}{2} \Delta x \) |
| Lax–Friedrichs \( \alpha_{\text{LF}} \) | 1.0 |

Table 2.: Parameter table for the one-dimensional FV-IPMM-method in case of the liquid–vapor flow multiscale model.
A Molecular–Continuum Multiscale Model for Inviscid Liquid–Vapor Flow with Sharp Interfaces

| FV-IPMM 2d |
|------------------|------------------|
| time step $\Delta t$ | $1 \cdot 10^{-4}$ |
| domain $\Omega$ | $(-1.5, 1.5)^2$ |
| max edge length $\Delta x$ | $1 \cdot 10^{-2}$ |
| motion regularization $\lambda_{motion}$ | $1 \cdot 10^{-4}$ |
| Lax–Friedrichs $\alpha_{LF}$ | 0.5 |

Table 3.: Parameter table for the two-dimensional FV-IPMM-method in case of the liquid–vapor flow multiscale model.

| FV-IPMM 3d |
|------------------|------------------|
| time step $\Delta t$ | $5 \cdot 10^{-4}$ |
| domain $\Omega$ | $(-1.5, 1.5)^3$ |
| max edge length $\Delta x$ | $3.5 \cdot 10^{-2}$ |
| motion regularization $\lambda_{motion}$ | $1 \cdot 10^{-4}$ |
| sliver threshold $\theta_{sliver}$ | 12° |
| Lax–Friedrichs $\alpha_{LF}$ | 0.5 |

Table 4.: Parameter table for the three-dimensional FV-IPMM-method in case of the liquid–vapor flow multiscale model.

References

[1] R. Abeyaratne and J. K. Knowles. “Kinetic relations and the propagation of phase boundaries in solids”. Arch. Ration. Mech. Anal. 114.2 (1991). doi: 10.1007/BF00375400.

[2] R. Abeyaratne and J. K. Knowles. Evolution of Phase Transitions: A Continuum Theory. Cambridge University Press, 2006. doi: 10.1017/CBO9780511547133.

[3] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. “On the Surprising Behavior of Distance Metrics in High Dimensional Space”. Database Theory — ICDT 2001. Ed. by J. Van den Bussche and V. Vianu. Springer, 2001. doi: 10.1007/3-540-44503-X.

[4] M. Alkämper and J. Magiera. Interface Preserving Moving Mesh (Code). Version v1.0. 2022. doi: 10.18419/darus-1671.

[5] M. Alkämper, J. Magiera, and C. Rohde. An Interface Preserving Moving Mesh in Multiple Space Dimensions. 2021. arXiv: 2112.11956 [cs.CG].

[6] M. P. Allen and D. J. Tildesley. Computer Simulation of Liquids. 2nd ed. Oxford University Press, Inc., 2017.

[7] D. A. Bezug, S. J. Schmidt, and N. A. Adams. “A data-driven physics-informed finite-volume scheme for nonclassical undercompressive shocks”. J. Comput. Phys. 437 (2021). doi: 10.1016/j.jcp.2021.110324.

[8] T. Blesgen. “A generalization of the Navier–Stokes equations to two-phase flows”. J. Phys. D: Appl. Phys 32.10 (1999). doi: 10.1088/0022-3727/32/10/307.

[9] G. Bussi, D. Donadio, and M. Parrinello. “Canonical sampling through velocity rescaling”. J. Chem. Phys. 126.1 (2007). doi: 10.1063/1.2408420.
[10] C. Chalons, J. Magiera, C. Rohde, and M. Wiebe. “A Finite-Volume Tracking Scheme for Two-Phase Compressible Flow”. *Theory, Numerics and Applications of Hyperbolic Problems*. 1st Ed. by C. Klingenberg and M. Westdickenberg. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-91545-6_25.

[11] C. Chalons, C. Rohde, and M. Wiebe. “A Finite Volume Method for Undercompressive Shock Waves in Two Space Dimensions”. *ESAIM: M2AN* 51.5 (2017). DOI: 10.1051/m2an/2017027.

[12] C. Chen and H. Hattori. “Exact Riemann solvers for conservation laws with phase change”. *Appl. Numer. Math.* 94 (2015). DOI: 10.1016/j.apnum.2015.03.008.

[13] R. M. Colombo and F. S. Priuli. “Characterization of Riemann solvers for the two phase p-system”. *Comm. Partial Differential Equations* 28.7-8 (2003). DOI: 10.1081/PDE-120024372.

[14] C. M. Dafermos. *Hyperbolic conservation laws in continuum physics*. 4th ed. Vol. 325. Springer, 2016. DOI: 10.1007/978-3-662-49451-6.

[15] W. E. *Principles of multiscale modeling*. Cambridge University Press, 2011.

[16] W. E, B. Engquist, X. Li, W. Ren, and E. Vanden-Eijnden. “Heterogeneous multiscale methods : A review”. *Comput. Phys. Commun.* 2.3 (2007).

[17] G. Faccanoni, S. Kokh, and G. Allaire. “Modelling and simulation of liquid-vapor phase transition in compressible flows based on thermodynamical equilibrium”. *ESAIM Math. Model. Numer. Anal.* 46.5 (2012). DOI: 10.1051/m2an/2011069.

[18] S. Fechter, F. Jaegle, and V. Schleper. “Exact and approximate Riemann solvers at phase boundaries”. *Comput. Fluids* 75 (2013). DOI: 10.1016/j.compfluid.2013.01.024.

[19] S. Fechter, C.-D. Munz, C. Rohde, and C. Zeiler. “A sharp interface method for compressible liquid–vapor flow with phase transition and surface tension”. *J. Comput. Phys.* 336 (2017). DOI: 10.1016/j.jcp.2017.02.001.

[20] A. Frezzotti and P. Barbante. “Simulation of shock induced vapor condensation flows in the Lennard–Jones fluid by microscopic and continuum models”. *Phys. Fluids* 32.12 (2020). DOI: 10.1063/5.0032439.

[21] H. Ghazi, F. James, and H. Mathis. “A nonisothermal thermodynamical model of liquid-vapor interaction with metastability”. *Discrete Contin. Dyn. Syst. Ser. B* 26.5 (2021). DOI: 10.3934/dcdsB.2020183.

[22] M. Hantke, W. Dreyer, and G. Warnecke. “Exact solutions to the Riemann problem for compressible isothermal Euler equations for two-phase flows with and without phase transition”. *Quart. Appl. Math.* 71.3 (2013). DOI: 10.1090/S0033-569X-2013-01290-X.

[23] M. Hantke and F. Thein. “On the Impossibility of First-Order Phase Transitions in Systems Modeled by the Full Euler Equations”. *Entropy* 21.11 (2019). DOI: 10.3390/e211111039.

[24] T. Hitz, S. Jöns, M. Heinen, J. Vrabec, and C.-D. Munz. “Comparison of macro- and microscopic solutions of the Riemann problem II. Two-phase shock tube”. *J. Comput. Phys.* 429 (2021). DOI: 10.1016/j.jcp.2020.110027.

[25] S. Homes, M. Heinen, J. Vrabec, and J. Fischer. “Evaporation driven by conductive heat transport”. *Mol. Phys.* 119.15-16 (2021). DOI: 10.1080/00268976.2020.1836410.

[26] J. H. Irving and J. G. Kirkwood. “The Statistical Mechanical Theory of Transport Processes. IV. The Equations of Hydrodynamics”. *J. Chem. Phys.* 18.6 (1950). DOI: 10.1063/1.1747782.
[27] J. Janeček. “Long Range Corrections in Inhomogeneous Simulations”. J. Phys. Chem. B 110.12 (2006). DOI: 10.1021/jp056344z.

[28] D. E. King. “Dlib-ml: A Machine Learning Toolkit”. J. Mach. Learn. Res. 10 (2009).

[29] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. 2014. arXiv: 1412.6980 [cs.LG].

[30] P. G. LeFloch and M. D. Thanh. “Nonclassical Riemann solvers and kinetic relations I. A nonconvex hyperbolic model of phase transitions”. ZAMP 52.4 (2001). DOI: 10.1007/PL00001563.

[31] P. G. LeFloch and M. D. Thanh. “Nonclassical Riemann solvers and kinetic relations. III. A nonconvex hyperbolic model for van der Waals fluids”. Electron. J. Differ. Equ. 72.1 (2000).

[32] P. G. LeFloch and M. D. Thanh. “Non-classical Riemann solvers and kinetic relations. II. An hyperbolic–elliptic model of phase-transition dynamics”. Proc. R. Soc. Edinb. A: Math. 132.1 (2002). DOI: 10.1017/S030821050000158X.

[33] T. Long, J. Cai, and S. Pan. “An accelerated conservative sharp-interface method for multiphase flows simulations”. J. Comput. Phys. 429 (2021). DOI: 10.1016/j.jcp.2020.110021.

[34] P. C. Ma, Y. Lv, and M. Ihme. “An entropy-stable hybrid scheme for simulations of transcritical real-fluid flows”. J. Comput. Phys. 340 (2017). DOI: 10.1016/j.jcp.2017.03.022.

[35] J. Magiera. “A Molecular–Continuum Multiscale Solver for Liquid–Vapor Flow: Modeling and Numerical Simulation”. PhD thesis. University of Stuttgart, 2021. DOI: 10.18419/opus-11797.

[36] J. Magiera. Data Sets for ’A Molecular–Continuum Multiscale Solver for Liquid–Vapor Flow: Modeling and Numerical Simulation’. Version V1. DaRUS, 2021. DOI: 10.18419/darus-1258.

[37] J. Magiera, D. Ray, J. S. Hesthaven, and C. Rohde. “Constraint-aware neural networks for Riemann problems”. J. Comput. Phys. 409 (2020). DOI: 10.1016/j.jcp.2020.109345.

[38] J. Magiera and C. Rohde. “A Particle-Based Multiscale Solver for Compressible Liquid–Vapor Flow”. Theory, Numerics and Applications of Hyperbolic Problems II. Ed. by C. Klingenber and M. Westdickenberg. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-91548-7_23.

[39] J. Magiera and C. Rohde. “Analysis and Numerics of Sharp and Diffuse Interface Models for Droplet Dynamics”. Droplet Dynamics under Extreme Ambient Conditions. Ed. by K. Schulte, C. Tropea, and B. Weigand. Fluid Mechanics and its Application. Springer, 2021. In print.

[40] M. Matsumoto. “Molecular dynamics of fluid phase change”. Fluid Ph. Equilibria 144.1 (1998). DOI: 10.1016/S0378-3812(97)00274-4.

[41] C. Merkle and C. Rohde. “The sharp-interface approach for fluids with phase change: Riemann problems and ghost fluid techniques”. ESAIM: M2AN 41.6 (2007). DOI: 10.1051/m2an:2007048.

[42] D. P. Mitchell. “Spectrally Optimal Sampling for Distribution Ray Tracing”. Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH ’91. ACM, 1991. DOI: 10.1145/122718.122736.

[43] J. Nickolls, I. Buck, M. Garland, and K. Skadron. “Scalable Parallel Programming with CUDA”. ACM Queue 6.2 (2008). DOI: 10.1145/1365490.1365500.
J. Magiera, C. Rohde

[44] S. Pan, L. Han, X. Hu, and N. A. Adams. “A conservative interface-interaction method for compressible multi-material flows”. *J. Comput. Phys.* 371 (2018). DOI: 10.1016/j.jcp.2018.02.007.

[45] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. *Advances in Neural Information Processing Systems* 32. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019.

[46] R. G. Patel, I. Manickam, N. A. Trask, M. A. Wood, M. Lee, I. Tomas, and E. C. Cyr. “Thermodynamically consistent physics-informed neural networks for hyperbolic systems”. *J. Comput. Phys.* 449 (2022). DOI: 10.1016/j.jcp.2021.110754.

[47] S. Popinet. “Numerical models of surface tension”. *Annual review of fluid mechanics*. Vol. 50. Annu. Rev. Fluid Mech. Annual Reviews, Palo Alto, CA, 2018.

[48] T. J. Santner, B. J. Williams, and W. Notz. *The design and analysis of computer experiments*. Vol. 1. Springer, 2003. DOI: 10.1007/978-1-4757-3799-8.

[49] R. Saurel, F. Petitpas, and R. Abgrall. “Modelling phase transition in metastable liquids: application to cavitating and flashing flows”. *J. Fluid Mech.* 607 (2008). DOI: 10.1017/S0022112008002061.

[50] V. Schleper. “A HLL-type Riemann solver for two-phase flow with surface forces and phase transitions”. *Appl. Numer. Math.* 108 (2016). DOI: 10.1016/j.apnum.2015.12.010.

[51] Y. Shen, Y. Ren, and H. Ding. “A 3D conservative sharp interface method for simulation of compressible two-phase flows”. *J. Comput. Phys.* 403 (2020). DOI: 10.1016/j.jcp.2019.109107.

[52] M. D. Thanh and D. X. Vinh. “The Riemann problem for van der Waals fluids with nonclassical phase transitions”. *Hokkaido Math. J.* 50.2 (2021). DOI: 10.14492/hokmj/2019-115.

[53] F. Thein. “Results for two phase flows with phase transition”. PhD thesis. Otto-von-Guericke-Universität Magdeburg, 2018. DOI: 10.25673/12275.

[54] M. Thol, G. Rutkai, A. Köster, R. Lustig, R. Span, and J. Vrabec. “Equation of State for the Lennard–Jones Fluid”. *J. Phys. Chem. Ref.* 45.2 (2016). DOI: 10.1063/1.4945000.

[55] A. N. Tichonov, A. S. Leonov, and A. G. Jagola. *Nonlinear Ill-posed problems*. Vol. 1–2. Chapman & Hall, 1998.

[56] L. Truskinovsky. “Kinks versus shocks”. *Shock induced transitions and phase structures in general media*. Vol. 52. IMA Volumes in Mathematics and its Applications. Springer, 1993. DOI: 10.1007/978-1-4613-8348-2_11.

[57] L. Verlet. “Computer Experiments on Classical Fluids. I. Thermodynamical Properties of Lennard–Jones Molecules”. *Phys. Rev.* 159.1 (1967). DOI: 10.1103/PhysRev.159.98.

[58] S. Yanagi. “The Riemann problem for a class of conservation laws of van der Waals fluid”. *Japan J. Indust. Appl. Math.* 9.2 (1992). DOI: 10.1007/BF03167567.

[59] C. Zeiler. “Liquid Vapor Phase Transitions: Modeling, Riemann Solvers and Computation”. PhD thesis. Universität Stuttgart, 2016. DOI: 10.18419/opus-8902.