A Case Study of AI-assisted Creation of a Thermodynamics Model of Precipitation Formation During Rapid Depressurization of a Vented Container

Jedediah M. Storey, Ph.D.
Thermal/Fluid Discipline Expert
NASA KSC Launch Services Program
Environments and Nuclear Launch Approval Branch

Artificial Intelligence (AI) Usage Disclosure

The model and code described in this document were created with assistance from ChatGSFC. The content has been reviewed by Jedediah Storey. More information on the extent of AI usage is included in the AI Case Study section. The document itself was not created with AI assistance.

Thermal & Fluids Analysis Workshop
TFAWS 2026
August 31 – September 4, 2026
NASA Marshall Space Flight Center
Huntsville, AL

Before we begin…

- AI was used in the development of the model but *not* to write this presentation or the accompanying paper.

- All opinions stated in this work are personal and are not the official position of NASA.

Introduction

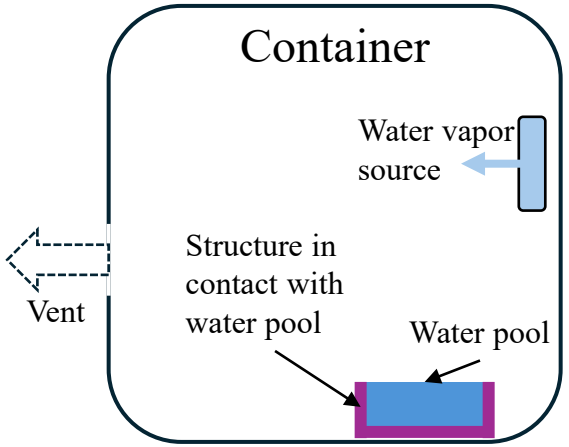
- Precipitation may form in humid containers undergoing rapid depressurization
  - Liquid (fog) if dewpoint > freezing
  - Ice (snow) if dewpoint < freezing
- Recent need to model this for a rapidly ascending vented container
  - Potentially relevant to vented containers/compartments in aircraft, spacecraft, and launch vehicles, as well as rapidly depressurizing vacuum chambers
- Anything that adds water vapor to container will enhance precipitation. Ex:
  - Water pool
    - Need to model convection, evaporation, boiling, freezing, sublimation, structure heat transfer, freezing while evaporating, and triple point boiling & freezing of the water pool
    - Transitions between regimes, e.g. L→B
  - Vapor sources such as foam or MLI

Objectives

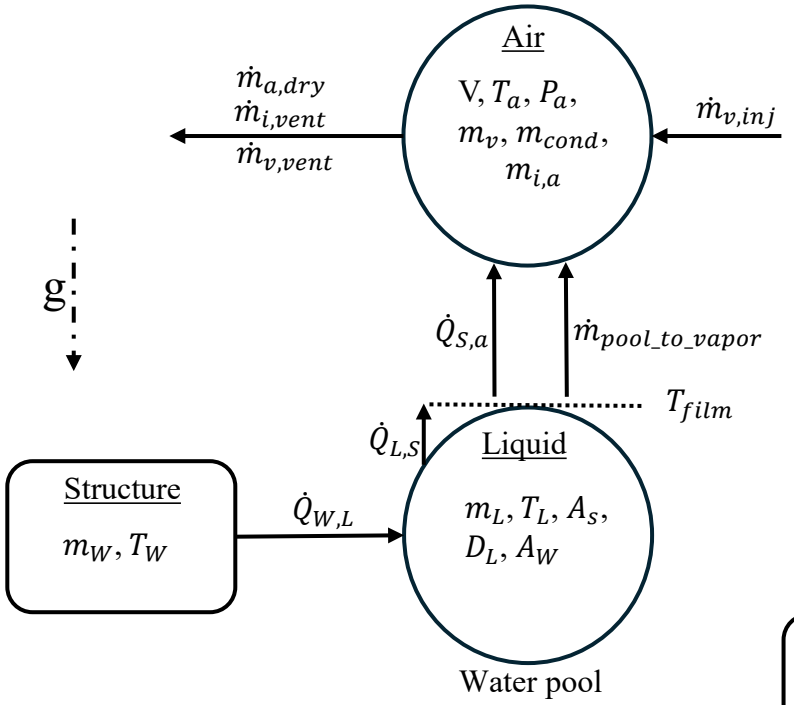
- Primary objective: to rapidly develop a thermodynamics model in python useful to engineers wishing to simulate rapidly depressurizing humid containers
- Secondary objective: to use an AI LLM to assist in its creation
- “Assist” = selection and development of physics models, code acceleration
- Model execution speed > maximum accuracy
- Objectives were met
- Math is in the paper
- Model built in python, ~10000 lines
- Will show a validation case and an example
- Will discuss experience using AI for this

- No novel modeling or thermodynamics methods

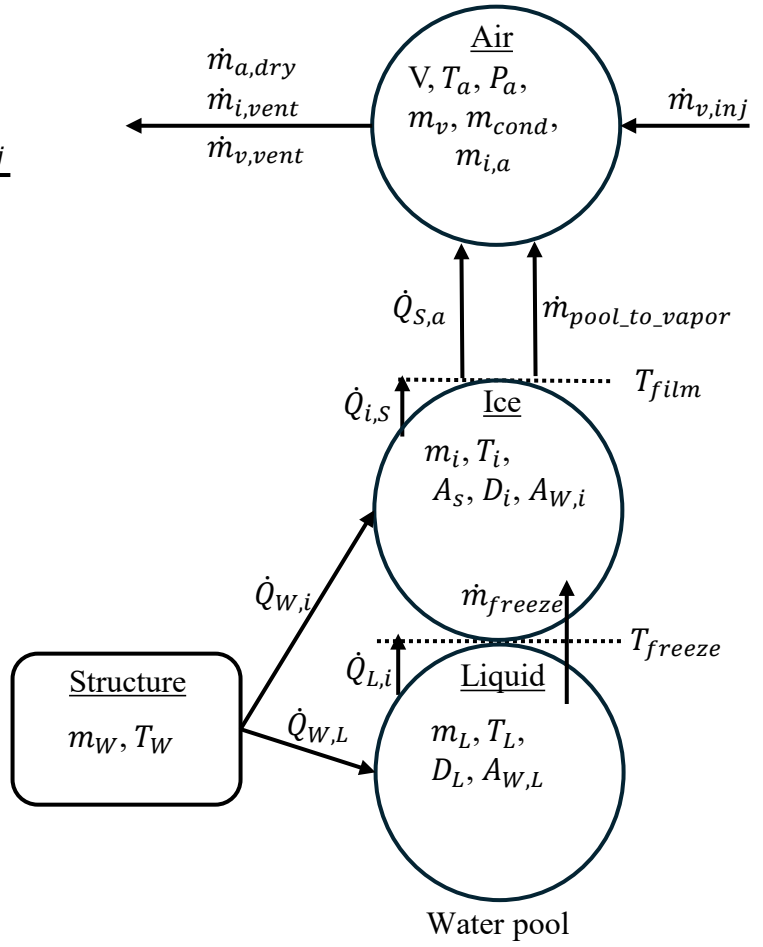
Model



Physical



Nodal



Notional physical representation and two nodal representations, one for a liquid pool, and one for a partially frozen pool.

Physics

- Container total air pressure prescribed. Container air temperature from polytropic expansion.
- Water vapor in the container air can condense/precipitate out as either liquid or ice. Excess water vapor is shunted to precipitate in each time step.
- Vented air is the composition of the container air: mix of air, water vapor, ice
- Optional Water Pool
  - Variable depth, area, volume pool
  - Convection: natural convection flat plate heat transfer correlations. Horizontal for interfaces, vertical for structure-pool heat transfer
  - Evaporation mass transfer from pool to air: Chilton-Colburn analogy
  - Low pressure (Kn<0.1) mass transfer: Hertz-Knudsen with user sublimation coefficient
  - Volumetric boiling only, no surface boiling. Calculated from heat balance
  - Boiling splash model
  - Freezing begins with a surface layer and progresses down
- Fluid properties from CoolProp and curve fits
- Some assumptions:
  - Perfectly mixed container air
  - Volume of container large enough that changes to the optional water pool volume don't affect container volume
  - Container walls have enough thermal mass to stay above dewpoint
  - Structure in contact with the pool has infinite thermal conductivity. A contact conductance is assumed.

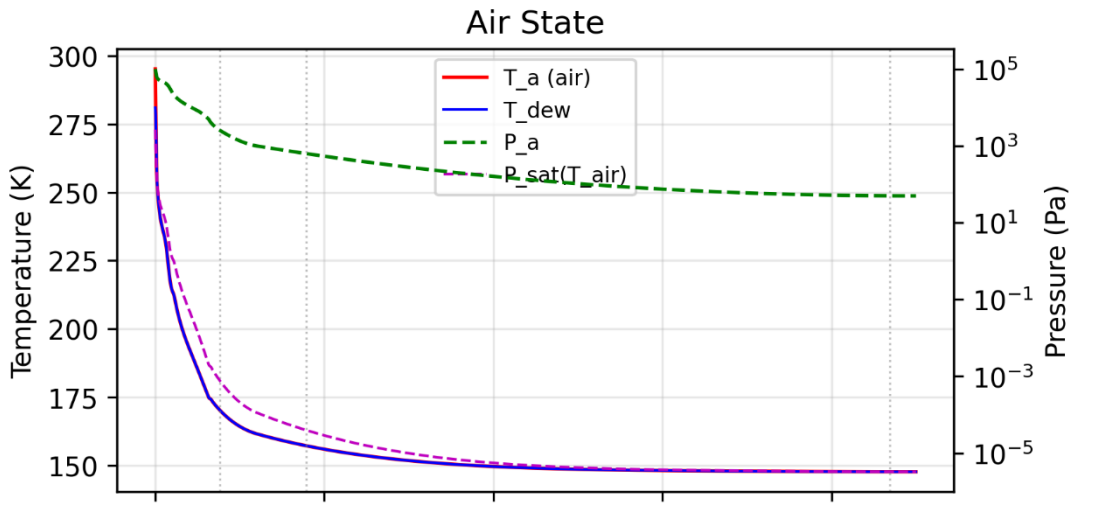
Code Structure

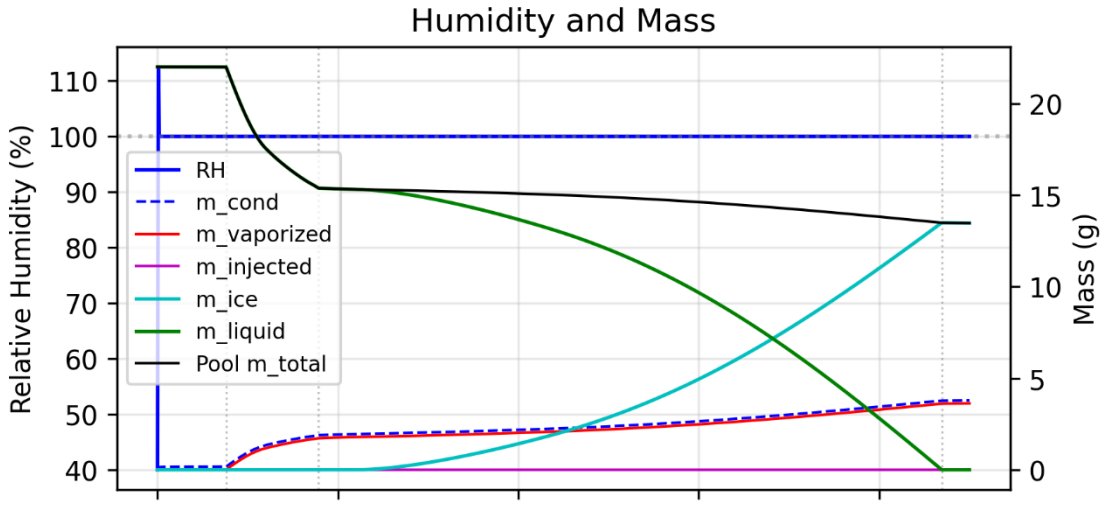
- Inputs:
  - container air pressure and temperature, g, geometry, initial state, user controlled tuning parameters
- Support modules: geom, air_data, props, vapor_sources, htc
- State variables: m_L, m_i, T_L, T_i, T_W, m_v, m_{cond}, $m_{i,vent}$
- Regime modules
  - Assembles the ODE system for the state variables in the current regime
  - L (liquid), LF (liquid freezing), B (boiling), TP (triple point), I (ice), D (dry)
  - regime_common: methods common to multiple regimes
  - transitions: smeared transitions between regimes while conserving mass and energy
- Driver module
  - scipy's solve_ivp 'BDF' is used to integrate the ODE system in time
  - "results" object stores the transient solution, derived parameters added to that after solve
  - Plotting function
- Execution time is typically less than a minute to a few minutes
- Instabilities

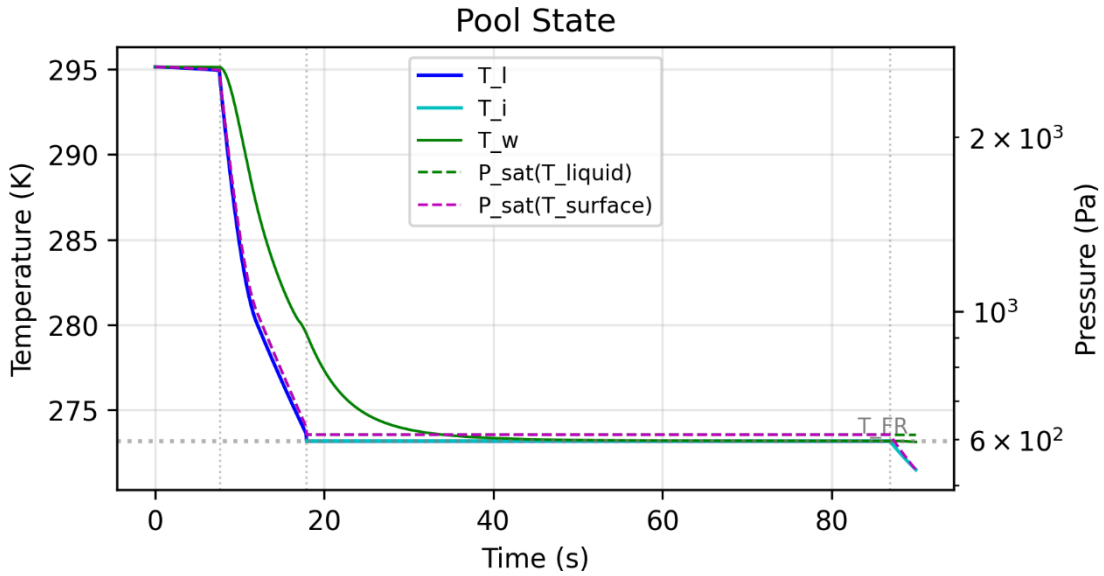
Validation Case

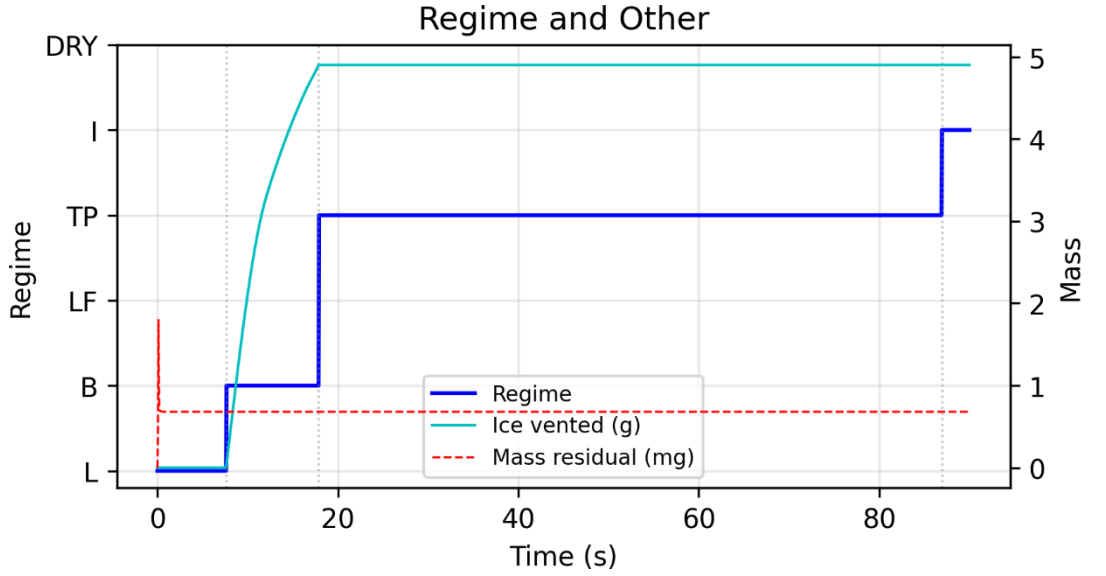
- https://www.youtube.com/watch?v=8SOCni8JfiU
- Standard 100 mL beaker with ~22 mL of water in it placed in a bell jar with attached evacuated chamber and large vacuum pump
- Valve is opened, near instant drop in pressure to ~60 kPa, then rapid drop to liquid saturation pressure in ~7.7 s, pool rigorously boils and some water splashes out, pressure continues to drop, surface starts freezing, then water completely freezes.
- Extracted estimates of everything needed to model this from video
  - Two video cuts, so took the “Water to ice in 90 seconds” in title literally
- Validation criteria:
  - Time boiling began (~7.7 s)
  - Most of time spent at triple point like in video?
  - Remaining ice mass (8 mm, ~13.3 g)

Validation Case: Simulation Results







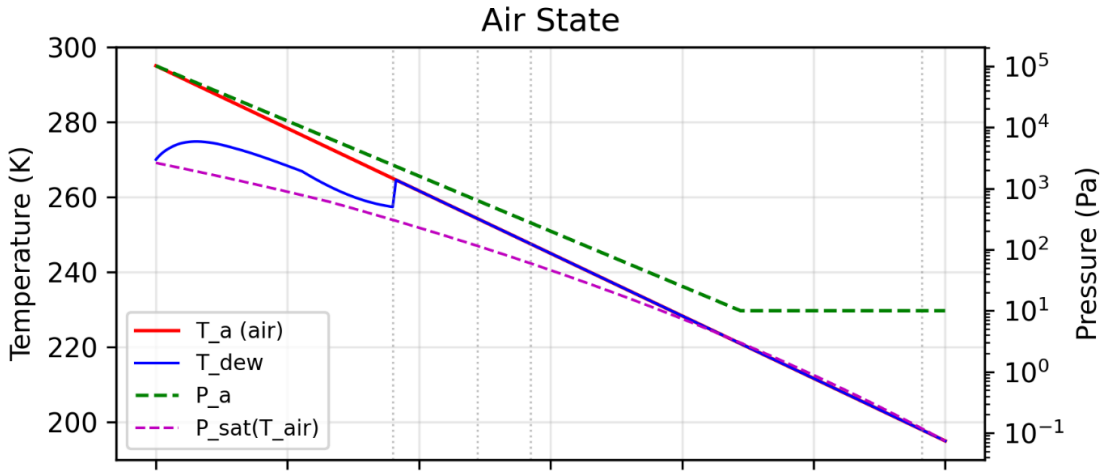


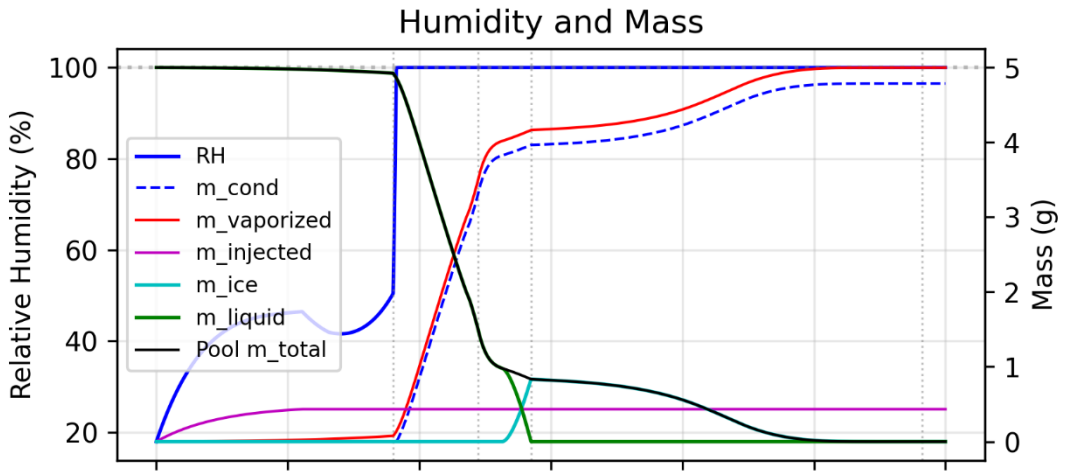
Example Case 1

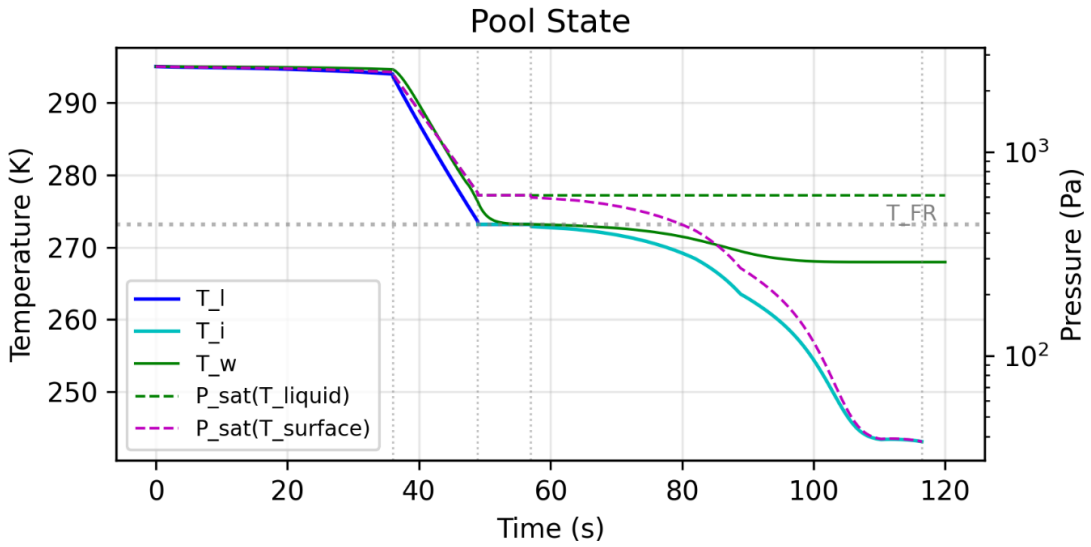
- Regimes: L→B→TP→I→D
- Inputs somewhat contrived to achieve desired regime changes.
- Rapid air pressure decay, linear temperature drop
- Pool is 1cm x 1cm x 0.5cm
- Includes a small water vapor injection source that shuts off at low pressure

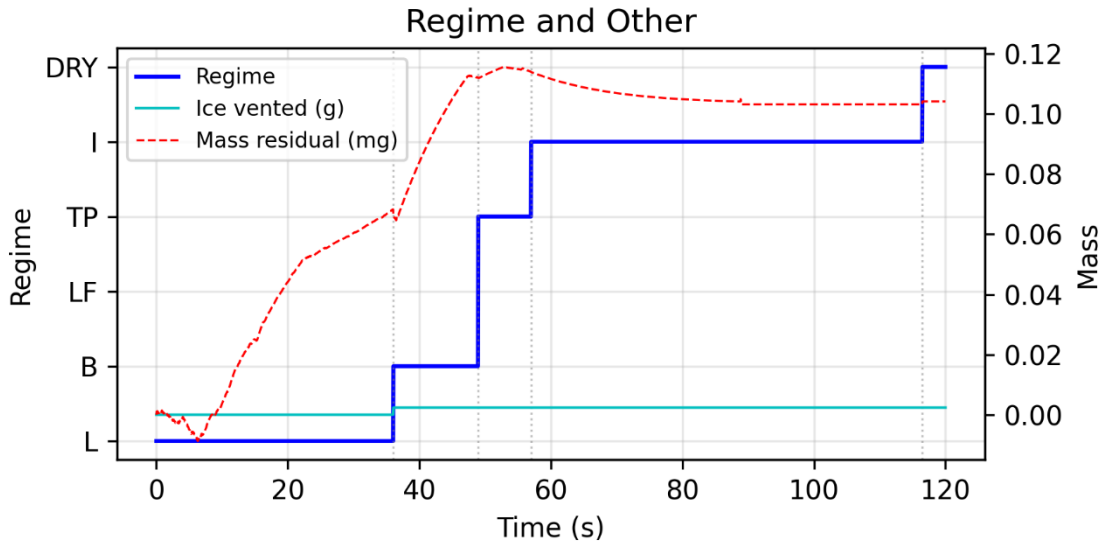
Parameter	Value	Units
max time	120	s
time step	0.05	s
V	0.05	m^3
$\boldsymbol{T_a}$	$295 - 50 * \left({}^{t}/_{60}\right)$	K
$\boldsymbol{P_a}$	$101325.0 * e^{(-\ln({}^{101325.0}/_{200}) * {}^{t}/_{60})}$, limited to 10	Pa
g	9.81	m/s^2
$\boldsymbol{T_{dew,init}}$	270	K
$\boldsymbol{T_{L,init}}$	295	K
$\boldsymbol{T_{W,init}}$	295	K
$\boldsymbol{m_{L,init}}$	5	g
$\boldsymbol{\dot{m}_{v,inj}}$	$0.00005 * \left({}^{P_a}/_{101325.0}\right)$ if $\boldsymbol{P_a} > 0.1 * 101325.0$, else 0.	kg/s
$\boldsymbol{A_{S,init}}$	0.01	m^2
$\boldsymbol{D_{L,init}}$	0.005	m
$\boldsymbol{A_{W,L,init}}$	0.04	m^2
$\boldsymbol{m_W}$	0.5	kg

Example Case 1









AI CASE STUDY

AI Overview

- ChatGSFC: Anthropic Claude Haiku 4.5 and Opus 4.7
- AI accelerated coding
- Wanted the AI to select correct physics and equations
- Tried Haiku first, then Opus
- Will discuss:
  - Strengths and weaknesses
  - Estimates of performance, time, and cost
  - Tips

Haiku

- Gradually added modeling complexity while providing code and physics descriptions.
- Asked the AI to defend its choices.
- AI could explain some physics correctly, such as the pressure differences between evaporation and volumetric boiling, but most of the physical insight came from me
- Misplaced constants inside the ODE loop, forgot wall heat transfer repeatedly, and repeatedly ignored instructions to use CoolProp when it was valid (Opus too…)
- AI sometimes gave partial modification instructions instead of directly modifying the code, and at one point forgot how to handle flash boiling despite having previously suggested the relevant formula
- Required user-guided debugging, AI self-debugging was ineffective
- Stability problems at regime transitions led to repeated troubleshooting loops
- AI was often confidently incorrect. It defended numerical artifacts as physical behavior, hallucinated sources, blamed already reverted changes, and messed up units math
- Later attempts using new chat windows, enthalpy-based reformulation, and a staged context-building approach produced some correct thermodynamic descriptions and equations and a partially functional code, but had the same stability problems and failed troubleshooting loops

Opus

- Suspected some issues with Haiku were due to it being a less capable LLM
- I prompted Opus with a description of the desired model and thermodynamics (without equations), asked for a detailed thermodynamics and heat-transfer description with equations, and explicitly told it not to write code yet
  - Despite requiring multiple “continue thinking” prompts due to ChatGSFC response context limitations, Opus produced a thorough and mostly correct response. Obviously superior to Haiku for this.
- Opus asked relevant clarification questions without being prompted
- Opus suggested laying out all possible regime transitions and associated state-variable mappings, then iterated on it and found additional edge cases without my input.
- When it felt like it had a complete-enough model description, it, unprompted, proposed a logical multi-module code structure
- We worked on one file at a time. Opus (unprompted) added unit tests to each file, which helped immensely with debugging, which it could usually do itself given the error output.
- Still had issues: it missed some loose solver tolerances, reverted to polynomial property fits instead of using CoolProp for some properties, made code changes that failed to produce desired demo-case modifications, and my idea is ultimately what fixed most of the stability issues.

Case Study Assessment

- Haiku:
  - Partial success, code partially worked.
  - Total time: ~30 hours
  - Total token cost: ~$4
  - I could have written something more functional in less time without it
- Opus:
  - Success. Met objectives of the project.
  - Total time: ~30 hours
  - Total token cost: ~$221, over 50x Haiku's, but still small compared to what my time costs
  - Wrote a better code than I could have in the same amount of time

Tips for Using LLMs to Develop Complex Physical Models

- Opus integrated into an IDE with the ability to run the code itself would be able to diagnose and fix errors. NASA has IDE AI tools available. NMC AI Hub (internal NASA Link)
  - This would also avoid the context limitations of ChatGSFC, which definitely hurt performance
- It is “lazy” and “forgetful”.
- It will lie to you.
- It will be extremely confidently wrong. “Incredibly knowledgeable idiot”
- Save versions of the code (use version control) so changes can be reverted easily if the AI breaks something or implements something that doesn’t work.
- It is useful for accelerating the writing of code.
- It can save time, but using it still requires a lot of time.
- Token costs are likely small compared to what the user’s time costs.
- It is good at formulating steps for “textbook” problem solutions.
- It has no comprehension of the physical world. The user needs to understand the physics.
- **It is not creative.** The user must supply the creative piece.

Summary and Future Work

- A functional thermodynamics model and corresponding python code useful to simulate rapidly depressurizing humid containers has been developed with the assistance of an AI LLM
- Flagship AI LLMs, such as Opus, may be useful when creating thermodynamics models, especially for accelerating the code development aspect. Cheaper LLMs are unlikely to be useful for tasks like this.
- Future model improvements:
  - Calculate the container air pressure given vent area, discharge coefficient, and transient external pressure for ascending containers, or alternatively, vacuum chamber/pump specifications. Then the container air temperature include heat transfer from the container walls and be treated as a state variable with an equation in the ODE system.
  - Several fluid properties are prescribed constants or use curve fits that could be partially replaced with CoolProp calls for improved accuracy.
  - More advanced boiling splash and convection enhancement models
  - Changing the heat transfer mechanism between the structure and pool as the pool becomes shallow would improve accuracy.

Questions?

The author would like to thank the NASA ChatGSFC team for their efforts developing a simple-to-use LLM interface.