Comment on egusphere-2025-2564', Simon O'Meara, 15 Jul 2025: Bezaatpour et al. present a digital tool (VaPOrS) for estimating pure component saturation vapour pressures, and from this property, the enthalpy of vaporisation. The properties discussed are fundamentally important to understanding aerosols, with significant implications for climate, weather and health. And it is welcome that efforts are being made to further our scientific understanding of this topic. I do hope the authors continue their important work in this area despite my review.

I have a fundamental concern with the submitted paper, which is that it makes an insubstantial contribution to modelling science, making its scientific significance too little to justify publication. Specifically, the referenced UManSysProp tool (Topping et al. 2016) already provides the vapour pressure estimation technique covered by VaPOrS. Then, we ask, do the tools differ significantly in their method to provide these properties? The authors demonstrate in their introduction that there is a variation in method, namely that whilst UManSysProp depends on the OpenBabel package to convert SMILES to SMARTS, which are then parsed, VaPOrS parses the SMILES directly. UManSysProp depends on a self-contained, human-defined, library of SMARTS to identify contributing groups (as described in and around Figure. 3 of Topping et al. 2016), whilst VaPOrS depends on a self-contained, (as far as I understand the paper, human-defined), library of SMILES to identify contributing groups. The Introduction of the paper argues that the VaPOrS method could give better control over pattern-matching logic than is possible in UManSysProp, however I can't see how this is true as both methods rely on a human to provide comprehensive libraries of relevant patterns (SMILES or SMARTS), and so the theoretical maximum degree of control is the same for both methods. Because this issue of insubstantial modelling significance is so important (justifying my rejection for publication) I do not provide further comments on other aspects of the paper at this stage.

Response:

We appreciate the reviewer’s concern regarding the novelty and significance of our contribution, particularly in relation to the existing UManSysProp tool. In our original manuscript, we deliberately chose not to emphasize direct comparisons with established methods in order to remain neutral and objective, aiming to allow users and modelers to evaluate tools based on their specific needs. However, in light of the reviewer’s comment

challenging the merits of VaPOrS relative to UManSysPro, we find it necessary to clarify and emphasize the methodological and practical strengths of our approach to defend its validity and utility. We respectfully submit that VaPOrS provides important advancements that directly address known limitations of UManSysPro, particularly in reading molecular structural information from a complex, general SMILES representation with subsequent estimation of condensational parameters that are critical for secondary organic aerosol (SOA) modeling.

1. Motivation from Practical Deficiencies in UManSysProp

While UManSysProp incorporates SMILES notations for vapor pressure estimation, our work originated from repeated, verifiable failures of UManSysProp in correctly identifying necessary functional groups in a wide range of organic species. These failures compromise the integrity of vapor pressure estimation, especially for chemically complex molecules relevant to atmospheric oxidation and SOA formation.

Specifically, we benchmarked UManSysProp and VaPOrS against the original compounds listed in the SIMPOL development paper (Pankow and Asher 2008). We found that UManSysProp failed to identify several functional groups essential for correct vapor pressure computation in some of these molecules (see Table 1). In contrast, VaPOrS correctly computed their vapor pressures in alignment with SIMPOL outputs.

Table 1. Benchmark of UManSysProp and VaPOrS against the original SIMPOL compounds from Pankow and Asher (2008). Several functional groups critical for vapor pressure estimation were missed by UManSysProp, resulting in deviations from SIMPOL values. VaPOrS accurately reproduced SIMPOL predictions in all cases. All vapor pressure values are expressed as log p (in atm).

Compound	SMILES	SIMPOL	UManSysProp	VaPOrS	Experimental
formamide	C(=O)N	-2.6493	1.8307	-2.6493	-3.1778
ethyl-formamide	CCNC=O	-3.2014	1.1292	-3.2014	-3.0353
methyl-formamide	CNC=O	-2.8507	1.4799	-2.8507	-2.6507
diethyl-formamide	CCN(CC)C=O	-2.0970	0.4278	-2.0970	-1.8235
dimethyl-formamide	CN(C)C=O	-1.3956	1.1292	-1.3956	-1.4152
dimethyl-hydroxylamine	CN(C)O	-1.1686	1.4799	-1.1686	-0.7208
n-butyl-benzoate	CCCCOC(=O)c1ccccc1	-3.3742	-2.2884	-3.3742	-3.4300
2-methyl-propyl-benzoate	CC(C)COC(=O)c1ccccc1	-3.3742	-2.2884	-3.3742	-2.9881

n-propyl-benzoate	CCCOC(=O)c1ccccc1	-3.0235	-1.9377	-3.0235	-2.7297
ethyl-benzoate	CCOC(=O)c1ccccc1	-2.6728	-1.5870	-2.6728	-2.5309
methyl-benzoate	COC(=O)c1ccccc1	-2.3220	-1.2363	-2.3220	-2.3216
acetic-acid,-phenyl-ester	CC(=O)Oc1ccccc1	-2.3220	-2.1088	-2.3220	-2.2995
dimethyl-1,2-benzenedicarboxylate	COC(=O)c1ccccc1C(=O)OC	-4.1093	-1.9377	-4.1093	-4.2168
dimethyl-benzene-1,3-dicarboxylate	COC(=O)c1cccc(c1)C(=O)OC	-4.1093	-1.9377	-4.1093	-4.1260
dimethyl-benzene-1,4-dicarboxylate	COC(=O)c1ccc(cc1)C(=O)OC	-4.1093	-1.9377	-4.1093	-4.0839
di-n-butyl-ethanedicarboxylate	CCCCOC(=O)C(=O)OCCCC	-3.4973	-1.3257	-3.4973	-3.3166
diethyl-ethanedicarboxylate	CCOC(=O)C=CC(=O)OCC	-2.7959	-0.7558	-2.7959	-2.5070
ethyl-2-nitropropionate	CCOC(=O)C(C)[N+](=O)[O-]	-3.3118	-2.4828	-3.3118	-2.6618
methyl-2-nitro-propionate	CC(C(=O)OC)[N+](=O)[O-]	-2.9611	-2.1320	-2.9611	-2.6133

In a broader evaluation, we assessed 126 primary VOCs provided by the Master Chemical Mechanism (MCM). UManSysProp produced incorrect vapor pressure estimates for at least four compounds (formaldehyde, formic acid, methyl ester, and BCARY) due to inaccurate group detection, while VaPOrS consistently matched the SIMPOL-based expectations (see Table 2).

Table 2. Evaluation of four primary VOCs from the MCM database. UManSysProp produced incorrect vapor pressure values due to the misidentification of functional groups, while VaPOrS matched SIMPOL outputs precisely. All vapor pressure values are expressed as log p (in atm).

Compound	**SMILES**	**SIMPOL**	**UManSysProp**	**VaPOrS**
formaldehyde	C=O	0.6863	1.8307	0.6863
formic acid	OC=O	-1.2397	1.8307	-1.2397
methyl ester	COC=O	0.3942	1.4799	0.3942
BCARY	C/C1=C/CCC(=C)C2CC(C)(C)C2CC\1	-3.5640	-3.4531	-3.5640

The implications become even more pronounced when analyzing oxidation products of major VOCs like benzene and α-pinene. For instance, UManSysProp failed to compute correct vapor

pressures for at least 20 α-pinene oxidation products (see Table 3) and 20 benzene oxidation products (see Table 4) due to missed functional groups. VaPOrS, on the other hand, successfully identified these groups and produced accurate results for all cases.

Table 3. Comparison of predicted vapor pressures for 20 α-pinene oxidation products. UManSysProp failed to recognize various multifunctional and peroxide-containing groups, leading to considerable errors. VaPOrS correctly identified all functional groups and reproduced the SIMPOL values. All vapor pressure values are expressed as log p (in atm).

SMILES	SIMPOL	UManSysProp	VaPOrS
[O]OC(=O)CC(=O)C=O	-2.2520	-1.3089	-2.2520
[O]OC(=O)C1CC(C(=O)O)C1(C)C	-4.7488	-3.8057	-4.7488
[O]OC(=O)CC1CC(C(=O)O)C1(C)C	-5.0995	-4.1564	-5.0995
[O]OC(=O)C1CC(C(=O)C)C1(C)C	-2.9721	-2.0290	-2.9721
O=CCC1CC(C(=O)[O])C1(C)C	-3.1733	-2.2302	-3.1733
O=CCC1CC(C(=O)O[O])C1(C)C	-3.1733	-2.2302	-3.1733
[O]OC(=O)CC1CC(C(=O)CO)C1(C)C	-5.1871	-4.2440	-5.1871
[O]OC(=O)CC1CC(C(=O)C)C1(C)C	-3.3229	-2.3797	-3.3229
CC(=O)O[O]	0.5368	1.4799	0.5368
[O]OC(=O)CC(=O)CC=O	-2.6027	-1.6596	-2.6027
[O]OC(=O)CC(=O)CC(=O)C(=O)C	-4.0461	-3.1030	-4.0461
[O]OC(=O)CC(=O)C(=O)C	-2.4015	-1.4584	-2.4015
CC(=O)C(O)C(=O)O[O]	-2.972	-2.0288	-2.972
[O]OC(=O)CC(=O)C(=O)CO	-4.2658	-3.3227	-4.2658
[O]OC(=O)CC(=O)CC(O)C(=O)C	-4.9673	-4.0241	-4.9673
CC(=O)C(O)CC(=O)O[O]	-3.3227	-2.3795	-3.3227
[O]OC(=O)CC=O	-0.9581	-0.0150	-0.9581
[O]OC(=O)C=O	-0.6074	0.3356	-0.6074
OCC(=O)O[O]	-1.3274	-0.3842	-1.3274
O=CC(=O)CC(=O)C(C)(ON(=O)=O)C(=O)O[O]	-6.1581	-5.2149	-6.1581

Table 4. Predicted vapor pressures of 20 benzene oxidation products. UManSysProp produced erroneous outputs due to functional group detection errors, particularly in conjugated and peroxide-bearing species. VaPOrS successfully identified all necessary groups and aligned with SIMPOL calculations. All vapor pressure values are expressed as log p (in atm).

SMILES	SIMPOL	UManSysProp	VaPOrS
[O]OC(=O)C1OC1C=CC=O	-2.7960	-1.8529	-2.7960

OC1COC(=O)C1=O	-3.2255	-2.1397	-3.2255
C1OC(=O)C=C1	-0.9785	0.1072	-0.9785
[O]OC(=O)C=CC(=O)C=O	-2.7342	-1.7911	-2.7342
OC(C=O)C(=O)C=CC(=O)O[O]	-4.9492	-4.0061	-4.9492
O=CCOC(=O)C=O	-2.5958	-1.5100	-2.5958
O=CCOC(=O)C(=O)O	-4.5220	-3.4362	-4.5220
[O]OC(=O)C1OC1C=O	-1.9631	-1.0200	-1.9631
[O]OC(=O)C=O	-0.6074	0.3356	-0.6074
OC(C=O)C(=O)O[O]	-2.8224	-1.8793	-2.8224
OCC(=O)O[O]	-1.3274	-0.3842	-1.3274
O=CC=CC(=O)[O]	-1.4403	-0.4972	-1.4403
[O]OC(=O)C=CC=O	-1.4403	-0.4972	-1.4403
O=CC(=CC(=O)[O])N(=O)=O	-3.2652	-2.3220	-3.2652
O=N(=O)C12OOC(C2O)C(O)([O])C(=C1)O	-8.2564	-6.3921	-8.2564
[O]OC1(O)C(=CC2(OOC1C2O)N(=O)=O)O	-8.2564	-6.3921	-8.2564
OOC1(O)C(=CC2(OOC1C2O)N(=O)=O)O	-10.3826	-8.5183	-10.3826
O=N(=O)OC1C2OOC1(C=C(O)C2([O])O)N(=O)=O	-8.3029	-6.4386	-8.3029
[O]OC1(O)C(=CC2(OOC1C2ON(=O)=O)N(=O)=O)O	-8.3029	-6.4386	-8.3029
OOC1(O)C(=CC2(OOC1C2ON(=O)=O)N(=O)=O)O	-10.4291	-8.5648	-10.4291

We further validated this in autooxidation studies. From recent high-resolution mass spectrometry data on the OH-initiated autooxidation of three aromatic carbonyls under high- and low-NOx conditions (Shawon et al.), we analyzed 180 identified oxidation products. UManSysProp failed to provide correct vapor pressure values for 67 of these compounds. VaPOrS, with direct SMILES-based group recognition, correctly handled every case (see Table 5).

Table 5. Analysis of oxidation products from OH-initiated autooxidation of aromatic carbonyls under different NOx conditions (from Shawon et al.). UManSysProp failed in 67 cases due to group detection limitations. VaPOrS handled all compounds correctly via explicit SMILES-based pattern recognition. All vapor pressure values are expressed as log p (in atm).

SMILES	**SIMPOL**	**UManSysProp**	**VaPOrS**
CC(=O)C(=O)C(O)=CC(=O)C(O)C=O	-8.4581	-6.5938	-8.4581
CC(=O)C(=O)C(O)=CC(=O)C(OO)C=O	-8.7200	-6.8557	-8.7200
CC(=O)C(=O)C(O)=CC(OO)C(=O)C=O	-8.7200	-6.8557	-8.7200
CC(=O)C(=O)C(O)=CC(O[O])C(O)C=O	-7.5149	-5.6506	-7.5149

CC(=O)C(=O)C(O)=CC(ON(=O)=O)C(O)C=O	-9.4257	-7.5614	-9.4257
CC(=O)C(=O)C(O)=CC(OO)C(O)C=O	-9.6411	-7.7768	-9.6411
CC(=O)C(=O)C(O)=CC(O)C(OO)C=O	-9.6411	-7.7768	-9.6411
CC(=O)C(=O)C(O)=CC(=O)C(O)C(=O)OO	-9.4298	-7.5655	-9.4298
CC(=O)C(=O)C(O)=CC(O[O])C(OO)C=O	-7.7768	-5.9126	-7.7768
CC(=O)C(=O)C(O)C(OO)C=CC(O[O])=O	-7.5756	-6.6325	-7.5756
CC(=O)C(=O)C(O)=CC(ON(=O)=O)C(OO)C=O	-9.6876	-7.8233	-9.6876
CC(=O)C(=O)C(O)=CC(O)C(O)C(=O)OO	-10.351	-8.4866	-10.351
CC(=O)C(=O)C(O)=CC(=O)C(OO)C(=O)OO	-9.6917	-7.8274	-9.6917
CC(=O)C(=O)C(O)=CC(O[O])C(O)C(=O)OO	-8.4866	-6.6223	-8.4866
CC(=O)C(=O)C(O)=CC(ON(=O)=O)C(O)C(=O)OO	-10.397	-8.5331	-10.397
CC(=O)C(=O)C(O)=CC(OO)C(OO)C(=O)O[O]	-9.7018	-6.8944	-9.7018
CC(=O)C(=O)C(O)=CC(O[O])C(OO)C(=O)OO	-8.7485	-6.8843	-8.7485
CC(=O)C(=O)C(O)=CC(OO)C(OO)C(=O)ON(=O)=O	-10.669	-8.8052	-10.669
CC(=O)C(=O)C(O)=CC(ON(=O)=O)C(OO)C(=O)OO	-10.659	-8.7950	-10.659
CC(=O)C(=O)C(O)=CC(OO)C(OO)C(=O)OO	-10.874	-9.0105	-10.874
[O]OCC(=O)C(=O)C(O)=CC(OO)C(O)C(OO)=O	-10.612	-8.7485	-10.612
O=N(=O)OCC(=O)C(=O)C(O)=CC(OO)C(O)C(OO)=O	-12.523	-10.659	-12.523
O=C(C=O)C(O)=CC(=O)C(O)C=O	-8.3085	-6.4442	-8.3085
O=C(C=O)C(O)=CC(=O)C(OO)C=O	-8.5704	-6.7062	-8.5704
O=C(C=O)C(O)=CC(OO)C(=O)C=O	-8.5704	-6.7062	-8.5704
O=C(C=O)C(O)=CC(O[O])C(O)C=O	-7.3654	-5.5011	-7.3654
O=C(C=O)C(O)=CC(O)C(OO)C=O	-9.4916	-7.6273	-9.4916
O=C(C=O)C(O)=CC(=O)C(O)OO	-8.9397	-7.0754	-8.9397
O=C(C=O)C(O)=CC(=O)C(O)C(=O)OO	-9.2802	-7.4159	-9.2802
O=C(C=O)C(O)=CC(O[O])C(OO)C=O	-7.6273	-5.7630	-7.6273
O=C(C=O)C(O)=CC(OO)C(OO)C=O	-9.7535	-7.8892	-9.7535
O=C(C=O)C(O)=CC(=O)C(OO)C(=O)OO	-9.5421	-7.6779	-9.5421
O=C(C=O)C(O)=CC(O[O])C(O)C(=O)OO	-8.3371	-6.4728	-8.3371
O=C(C=O)C(O)=CC(O[O])C(OO)C(=O)OO	-8.5990	-6.7347	-8.5990
O=C(C=O)C(O)=CC(OO)C(OO)C(=O)OO	-10.725	-8.8609	-10.725
O=CCC(=O)C(O)=CC(=O)C(O)C=O	-8.659	-6.7950	-8.659
O=CCC(=O)C(O)=CC(=O)C(OO)C=O	-8.9212	-7.0569	-8.9212
O=CCC(=O)C(O)=CC(OO)C(=O)C=O	-8.9212	-7.0569	-8.9212
O=CCC(=O)C(O)=CC(O[O])C(O)C=O	-7.7161	-5.8518	-7.7161
O=CCC(=O)C(O)=CC(ON(=O)=O)C(O)C=O	-9.6269	-7.7626	-9.6269
O=CCC(=O)C(O)=CC(O)C(OO)C=O	-9.8423	-7.9780	-9.8423
O=CCC(=O)C(O)=CC(=O)C(O)C(=O)OO	-9.631	-7.7667	-9.631
O=CCC(=O)C(O)=CC(O[O])C(OO)C=O	-7.9780	-6.1137	-7.9780
O=CCC(=O)C(O)=CC(ON(=O)=O)C(OO)C=O	-9.8888	-8.0245	-9.8888
O=CCC(=O)C(O)=CC(OO)C(OO)C=O	-10.104	-8.2399	-10.104
O=CCC(=O)C(O)=CC(O[O])C(O)C(=O)OO	-8.6878	-6.8235	-8.6878
O=CCC(=O)C(O)=CC(ON(=O)=O)C(O)C(=O)OO	-10.598	-8.7343	-10.598

O=CC(=O)C(=O)C(O)=CC(OO)C(O)C(=O)OO	-11.757	-9.8929	-11.757
O=CCC(=O)C(O)=CC(OO)C(OO)C(=O)O[O]	-9.9030	-7.0956	-9.9030
[O]OC(C=O)C(=O)C(O)=CC(OO)C(OO)C=O	-10.104	-8.2399	-10.104
O=CCC(=O)C(O)=CC(O[O])C(OO)C(OO)=O	-8.9497	-7.0854	-8.9497
O=CCC(=O)C(O)=CC(OO)C(OO)C(=O)ON(=O)=O	-10.870	-9.0064	-10.870
O=N(=O)OC(C=O)C(=O)C(O)=CC(OO)C(OO)C=O	-12.015	-10.150	-12.015
O=CCC(=O)C(O)=CC(ON(=O)=O)C(OO)C(OO)=O	-10.860	-8.9962	-10.860
O=CCC(=O)C(O)=CC(OO)C(OO)C(=O)OO	-11.076	-9.2116	-11.076
O=CC(OO)C(=O)C(O)=CC(OO)C(OO)C=O	-12.230	-10.366	-12.230
O=CC(O[O])C(=O)C(O)=CC(OO)C(O)C(=O)OO	-10.814	-8.9497	-10.814
O=CCC(=O)C(=O)C(OO)C(OO)C(O)C(=O)O[O]	-10.714	-9.7716	-10.714
O=CC(ON(=O)=O)C(=O)C(O)=CC(OO)C(O)C(=O)OO	-12.724	-10.860	-12.724
[O]OC(=O)C(OO)C(=O)C(O)=CC(OO)C(OO)C=O	-12.029	-9.2218	-12.029
O=CCC(=O)C(=O)C(OO)C(OO)C(OO)C(=O)O[O]	-10.976	-10.033	-10.976
O=N(=O)OC(=O)C(OO)C(=O)C(O)=CC(OO)C(OO)C=O	-12.996	-11.132	-12.996
OOC(=O)C(OO)C(=O)C(O)=CC(OO)C(OO)C=O	-13.202	-11.337	-13.202
O=CCC(=O)C(O)=CC(C(O)C=O)OOOC(C=O)C=CC(O)C(=O)CC=O	-15.749	-13.885	-15.749
O=CCC(=O)C(O)=CC(C(O)C=O)OOOC1C=CC2OOC1C2(O)CC=O	-14.446	-12.582	-14.446
O=CCC(=O)C(O)=CC(C(O)C=O)OOC(C(O)C=O)C=C(O)C(=O)CC=O	-18.176	-14.447	-18.176
O=CCC(=O)C(O)=CC(C(OO)C=O)OOC(C(OO)C=O)C=C(O)C(=O)CC=O	-18.700	-14.971	-18.700

The challenges identified above are not isolated cases but represent systemic limitations of the existing tool when applied to chemically diverse and rapidly growing databases of atmospheric oxidation products. With new high-resolution mass spectrometry findings continually introducing thousands of new molecules into atmospheric models, a tool like VaPOrS that offers reliable, transparent, scalable, and portable group detection and vapor pressure estimation becomes a valuable contribution to the field.

2. Methodological Differences and Computational Efficiency

VaPOrS provides **full control and transparency** over the patterns and matching logic used for functional group identification and vapor pressure calculation directly from SMILES strings, making the process fully auditable and easily modifiable. In contrast, the internal operations of Open Babel used by UManSysPro via its Python wrapper Pybel, are less transparent. Modifying behavior within Open Babel typically requires deep expertise in C++ and SWIG (Simplified Wrapper and Interface Generator), which imposes a steep learning curve on non-

core developers to extend or troubleshoot functional group detection [1-3]. Users of UManSysPro have limited visibility into or control over these pattern definitions, complicating diagnosis of missed functional groups or extending detection logic without advanced knowledge of the underlying cheminformatics library.

The reliance on third-party libraries (Open Babel and Pybel) also introduces **computational overhead** in UManSysPro because Pybel converts SMILES strings into internal molecular graph objects before querying functional groups. This step is heavier computationally compared to direct pattern matching on SMILES strings. In large-scale atmospheric secondary organic aerosol modeling scenarios, where thousands of species are processed across thousands of time steps and spatial grid cells, this overhead multiplies substantially, leading to significant decreases in performance. By bypassing the need for Open Babel and Pybel, **VaPOrS can reduce runtime dramatically at atmospheric modeling scales**. For example, with 100 compounds, 10,000 spatial grid cells, and 10,000 time steps, equating to approximately 10^{10} vapor pressure calculations, even saving as little as 1 millisecond per call yields an overall runtime reduction on the order of days to weeks of CPU time. This order-of-magnitude speedup exemplifies the practical computational advancement that direct SMILES parsing and pattern matching, as implemented in VaPOrS, can provide over UManSysPro's Pybel-dependent approach.

Even though VaPOrS's Pybel-free Python code performs much faster than UManSysPro's Pybel-based implementation, it remains subject to **Python interpreter overhead** and suboptimal scaling compared to compiled languages. Fortran, for example, is the language of choice for most leading-edge high-performance simulation codes such as PALM [4] and ADCHEM [5], due to its superior efficiency in large-scale numerical computations and direct memory management [6]. Direct comparisons show Fortran can be hundreds of times faster than pure Python for the same algorithm. For example, a realistic computational task took 66 seconds in Fortran versus 33,900 seconds (9.5 hours) in Python, a factor of 500 difference [7]. Fortran manages memory more efficiently, reducing runtime and minimizing memory exhaustion risks compared to Python's dynamic and object-heavy environment. Modern Fortran, with support for OpenMP and MPI, is well suited for parallelizing across multi-core CPUs and HPC (high-performance computing) clusters, ideal for the demanding grids and time steps of atmospheric SOA modeling. Compiled Fortran code runs consistently across platforms, avoiding variability caused by the Python interpreter and dynamic dependency issues. Regarding this, UManSysPro's dependency on Open Babel/Pybel creates challenges for

portability and maintainability. Open Babel is a large C++ library with Python bindings, and porting its entire dependency chain to other programming languages, especially Fortran, is impractical and cumbersome. The typical solution, such as writing complex wrappers or using foreign function interfaces, is technically challenging and fragile. Fortran's limited interoperability with C++ libraries without extensive tooling contrasts sharply with VaPOrS's algorithmic approach, which is based solely on standard string operations and pattern matching, making it **straightforward to port to Fortran**. Furthermore, calling UManSysPro, which uses Python, from the above-mentioned Fortran-based models for vapor pressure calculations at every time step and grid cell for many compounds introduces a **mixed-language dependency that can incur significant runtime overhead**. This overhead arises from interpreter invocation and data marshalling between Fortran and Python, **cumulatively adding up to the slowdown** already caused by Pybel.

If the vapor pressure calculation code is ported natively into Fortran, **as VaPOrS's algorithmic simplicity allows straightforward translation to Fortran**, these penalties vanish. Fortran's compiled nature allows **massively faster execution**, seamless integration, and optimized parallelism, critical for atmospheric scale modeling. Pure numerical and string logic implemented in Fortran will maximize CPU throughput and scale smoothly across large model domains without bottlenecks imposed by language design. Therefore, VaPOrS's practical and potential advantages in reducing runtime can be summarized as follows:

- **Avoiding unnecessary computations** by bypassing Pybel's SMILES-to-graph conversion and using direct pattern matching.
- **Eliminating Python–Fortran interface overhead** by avoiding mixed-language calls during large-scale simulations.
- **Allowing pure Fortran translation**, enabling much faster native execution compared to Python.

3. Execution Conveniency

Running UManSysPro on supercomputers (high-performance computing, HPC) can lead to several complainable problems, especially in large-scale modeling workloads typical of atmospheric science. Supercomputers frequently run custom or stripped-down Linux environments. Installing complex dependencies like Open Babel and its Python wrappers

(Pybel, SWIG bindings) is often problematic. Users may need special compilation steps, manual resolution of C++/Python/Swig compatibility, or may encounter issues with missing or incompatible shared libraries. Open Babel/Pybel and all dependent Python packages must be installed and maintained across all compute nodes. Minor discrepancies in versions or build environments between nodes can cause errors that are very hard to track in distributed jobs, leading to wasted supercomputing allocations. Furthermore, some supercomputer environments restrict the installation of non-standard libraries or require jobs to run only using centrally installed software. Getting approval or support for Open Babel/Pybel can be challenging, especially when the core libraries are actively developed with potentially breaking changes [8-10]. These factors are widely acknowledged as major barriers to deploying Python-based, C++-linked scientific tools like UManSysPro on supercomputers compared to simpler, compiled, single-language scientific code. These issues often motivate writing more portable, lightweight, and easily compilable tools for large HPC applications, or at least favoring solutions that can be run natively in Fortran on all target architectures without Python or complex bindings.

Conclusion

While both UManSysProp and VaPOrS nominally implement the SIMPOL method, the pathway to functional group identification and vapor pressure computation is substantially different. VaPOrS's direct SMILES parsing avoids known limitations of Pybel-based systems, offers a more flexible and extensible framework, and has demonstrably outperformed UManSysProp across multiple atmospheric datasets, including benchmark compounds, MCM species, oxidation products, and autooxidation products.

We hope this detailed explanation addresses the reviewer’s concern regarding the significance and novelty of our approach and demonstrates that VaPOrS represents a robust advancement in modeling vapor pressure for atmospheric applications.

References

1. Write software using the Open Babel library - Read the Docs.
2. Open Babel - Docs CSC.
3. Python wrapper for the OpenBabel cheminformatics toolkit - PMC.
4. Overview of the PALM model system 6.0 - GMD.
5. ADCHEM | Multi-Scale Modelling | University of Helsinki
6. The counter-intuitive rise of Python in scientific computing.
7. A python vs. fortran smackdown - Guy Worthey.
8. https://bugs.archlinux.org/task/21708.
9. https://stackoverflow.com/questions/77224942/installing-openbabel-for-python-isnt-working.
10. https://sourceforge.net/p/openbabel/mailman/openbabel-discuss.