# Advantages of ChimeraX Phenix integration

Phenix is arguably the leading suite of tools (by number of citations) for crystallographic and cryo-EM model-building and refinement in use today. It incorporates a wide range of tools for experimental phasing and molecular replacement (that is, determination of the basic “layout” of a new crystal structure), automated model building (extension of a starting model into the density based on known sequence), analysis (e.g. determination of arbitrary symmetry in a cryo-EM map, detection of pathologies in crystallographic data), refinement and comprehensive validation.

What Phenix has always lacked is a well integrated graphical front-end for model visualisation and human-driven rebuilding. Here it relies on *Coot*, developed as part of the CCP4 suite. Ultimately, I would like to see ChimeraX and ISOLDE fill this role – not necessarily replacing *Coot*, but filling in the spaces where it has historically struggled (high quality visualisation in terms of ChimeraX’s strengths; maintaining model quality at low resolution in terms of ISOLDE’s).

The existing interface between Phenix and *Coot* is very one-directional: Phenix refines a model and shows interim results in *Coot* via a simple XMLRPC interface, but there is no communication in the other direction: *Coot* uses nothing from Phenix, and the only way for the user to run a fresh refinement after rebuilding is to save their model to a file and re-open it via the Phenix GUI.

I can see many advantages to a much tighter bi-directional integration between Phenix (and its underlying open-source library CCTBX) and ChimeraX, whether at arm’s length via a REST or socket-based interface or (more powerful, but significantly more difficult) by running both packages in the same Python environment. This latter approach historically has not been possible due to Phenix being based on Python 2.7, but after a great deal of work over the last year the migration to Python 3.6-8 (and PySide2 GUI) is nearly complete. Future versions of CCTBX and Phenix will be based on the Conda framework, which should make integration more straightforward than it was before.

Specific advantages I can see include:

- integration with Phenix’s auto-building tools for a faster and (optionally) more interactive model building pipeline. These tools are largely based in fragment-based matching to traces through density. While quite powerful (particularly at resolutions better than about 3 Angstroms) they become very slow (in the order of hours to days for a large model) when aiming for maximum completeness, and begin to fail badly beyond 3 Angstroms. Major reasons for this slowness that I can see are (a) the need to generate and then judge multiple mutually-exclusive paths; (b) difficulties in matching sidechains to density, particularly for larger and more flexible sidechains in lower-resolution density; and (c) poor geometry where fragments join. My initial experiments suggest that including ISOLDE (whether human-driven or automatic) in the build-refine-judge-build cycle can help enormously here: it tends to very rapidly pull somewhat-wonky-but-essentially-correct segments into better conformations (and fit) while avoiding most of the force-fitting of truly-wrong spots that can happen with traditional crystallographic restraints. My essential vision here would be a tool where the user defines a start and end point for a missing fragment, an associated sequence and a box of density, and asks Phenix to do its best using its “quick” mode (this is quite fast, on the order of seconds for a scenario like this). The user then gets the opportunity to correct what has been built (settling/remodelling into density, trimming away anything completely wrong) before iterating again. Done properly, this alone could be extremely powerful, significantly easing and accelerating what is arguably the most time-consuming step in building novel proteins.

- Phenix’s crystallographic map calculations (based on CCTBX) are significantly more advanced and rigorous than mine (using Clipper). They are, however, much slower at present. Part of this is simply because the core C++ component is not parallelised (its single-threaded performance is actually about 30% faster than Clipper for the same problem) – Duncan Stockwell in the Read lab is currently looking at fixing this. The remainder is inefficiencies in the Python layer which could be mitigated/avoided with some further work. If it were possible, switching from Clipper to CCTBX for handling of crystallographic maps and symmetry would make things much easier both practically and politically. Practically, in the sense that ISOLDE would automatically benefit from the 20-30 years of theoretical advances in the field embodied in CCTBX. Politically, because my current path will otherwise put me in effective competition with Phenix: my overall vision with ISOLDE is essentially to merge model building, refinement and validation into one smooth continuous process – which, taken to its logical extreme, would eventually take “current” Phenix out of the picture entirely. I would much prefer to avoid that conflict if possible, and joining forces seems the most obvious way to do so. If I could have used CCTBX when starting this project I would have, but the incompatibility of Python versions made that effectively impossible (and the size and complexity of the code base is far too daunting for me to have taken on the task of modernising it). There is a down-side, though: while CCTBX is undoubtedly more advanced than Clipper in its capabilities it is *much* harder to understand and sparse on documentation (or inline code commenting), particularly in its C++ layer.

- Airlie McCoy (here in the Read Lab, primary developer of *Phaser* for molecular replacement) has often talked about wanting to look seriously at ChimeraX as a potential front-end. This (I think) would be a relatively straightforward task compared to anything to do with model building, primarily involving showing the same model in a range of possible poses along with the resulting maps. There may be some scope for help from ISOLDE here as well: while molecular replacement is mostly a rigid-(multi)body search, once you get to the point where solutions are looking sufficiently “real” then some tightly-restrained settling of the model into the map may be justified.

- One thing that ISOLDE doesn’t do right now is refine atomic B-factors. This has no real practical impact for fitting of cryoEM maps (but is important for *judging* of fit to density and for final interpretation of the model), but has a serious impact in crystallography where errors in B-factors directly affect the quality of the calculated map. This again is an enormously complicated field; it would be great to be able to work directly with the refinement algorithms already implemented in Phenix. While I believe these do have room for improvement at low resolutions, this would save a huge amount of effort (and political capital) that would need to be spent in reimplementation, and set a solid base for future improvement.

- Phenix (specifically, the command-line tool *phenix.elbow*) now incorporates an ANTECHAMBER-based pipeline for ligand parameterisation, which would provide an easy route to supporting most novel ligands in ISOLDE. Current limitations are that it only supports non-covalently bound ligands, and fails on metal-containing systems (e.g. heme). Like all Phenix command-line tools, *phenix.elbow* is a wrapper around a Python script. Going forward, the plan is for all of these tools to be based on a common template class (designed by Billy Poon on the Phenix team), providing a fairly consistent API. This would probably be an easy sell funding-wise in the context of the current COVID-19 crisis – I expect over the coming months we’re going to see a flood of complexes with potential inhibitors (experimentally determined and computationally predicted alike).

## REST Interface

I have an initial, working implementation of how I envisage a REST-based interface between ChimeraX and ISOLDE. The code is at <https://github.com/tristanic/isolde/tree/master/isolde/src/remote_control/rest_server>. The server side can be started with the ChimeraX command “isolde remote rest start port 12345”. The file client.py is a standalone file that should be importable in any Python 2.7 or >=3.6 environment. With ChimeraX running the server, the client connects using something like:

from client import IsoldeRESTClient

port = 12345

ic = IsoldeRESTClient(‘localhost’, port)

ic.connect() # Client class becomes populated with methods defined

# by the server. These appear as normal Python

# methods, with the proviso that arguments must be

# JSON-serialisable. The return type is a dictionary

# that always contains the log contents plus any

# method-specific information.

The currently defined server methods are listed at [https://github.com/tristanic/isolde/blob/5f65a6f0f958ff23fec0ff81a0c9933f95d0edf5/isolde/src/remote\_control/\_\_init\_\_.py#L41](https://github.com/tristanic/isolde/blob/5f65a6f0f958ff23fec0ff81a0c9933f95d0edf5/isolde/src/remote_control/__init__.py" \l "L41) (the actual method definitions are in <https://github.com/tristanic/isolde/blob/master/isolde/src/remote_control/server_methods.py>). The methods reconstituted on the client side are based on introspection – they have the same signature (minus the initial `session` argument) and docstring.

As an example, continuing on from above:

ret1 = ic.load\_model(‘full/path/to/model.cif’)

mid = ret2[‘model’]

ret2 = ic.load\_stucture\_factors(‘full/path/to/sf.mtz’, mid)

mid = ret2[‘model’] # Shouldn’t have actually changed…

ret3 = ic.update\_model\_from\_file(model, ‘full/path/to/new\_model.cif’)

# Closes the existing model, and replaces it with a fresh one

While incomplete, the methods defined here already go a long way towards replicating the existing remote interface between Phenix and *Coot*, with the improvement that this server actually returns information beyond “success” and “failure”. An obvious extension allowing easy “arms-length” communication between ChimeraX and Phenix would be to run a similar server on the Phenix side allowing ChimeraX to remotely call selected Phenix methods. Another fairly easy improvement might be to switch from JSON to the much more flexible msgpack (I didn’t know about the latter when I wrote this).