MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Modern visualization of partial atomic charges in Mol*

Bachelor's Thesis

DOMINIK TICHÝ

Brno, Spring 2023

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Modern visualization of partial atomic charges in Mol*

Bachelor's Thesis

DOMINIK TICHÝ

Advisor: RNDr. Tomáš Raček, Ph.D.

Department of Computer Systems and Communications

Brno, Spring 2023


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

I declare that I have worked on this thesis independently, using only the primary and secondary sources listed in the bibliography.

Dominik Tichý

Advisor: RNDr. Tomáš Raček, Ph.D.

Acknowledgements

TODO

Abstract

TODO

Keywords

Molstar, Mol*, Atomic Charge Calculator 2, ACC2, AlphaCharges, αCharges, SB NCBR, molecular visualization, molecular graphics, partial atomic charges, structural biology, AlphaFold

Contents

List of Tables

List of Figures

Introduction

1 Theory

TODO: this chapter is has placeholder text and is not finished yet.

Theoretical concepts are fundamental to the study of computational chemistry, providing a framework for analyzing molecular structures and properties. This chapter focuses on four key areas of theory, beginning with an overview of molecular structure in Section 1.1. Section 1.2 provides an in-depth examination of chemical file formats, including the advantages and disadvantages of different file formats for storing molecular data. Partial atomic charges are explored in section 1.3, including their importance in analyzing molecular structures and the various methods used to compute them. Finally, section 1.4 delves into color interpolation, a critical technique for visualizing partial atomic charges in molecular structures.

By providing a comprehensive overview of these theoretical concepts, this chapter provides a strong foundation for the subsequent chapters, focusing on the implementation and analysis of the Mol* extension for visualizing partial atomic charges in molecular structures.

1.1 Molecular structure

TODO: the section is not split into many subsections to allow for better reading flow; first i describe small molecules (micromolecules) which are just described by their atoms and bonds, these are usually drug molecules (think ligands, ions etc); for macromolecules explain the hierarchy of chain-residue-atom, in this work we used carbohydrates and proteins - describe those as well; explain somehow that this separation is done on purpose; add a schematic image that shows a small protein with 2 chains, then zoom in on one chain and show the residues, lastly zoom on one of the residues to show the atoms and bonds

TODO: don't focus too much on explaining things that aren't relevant further in the thesis

1.1.1 Micromolecules

1.1.2 Macromolecules

Molecular structure refers to the arrangement of atoms and chemical bonds in a molecule. The three main components of molecular structure are atoms, residues, and chains. In this section we will look at

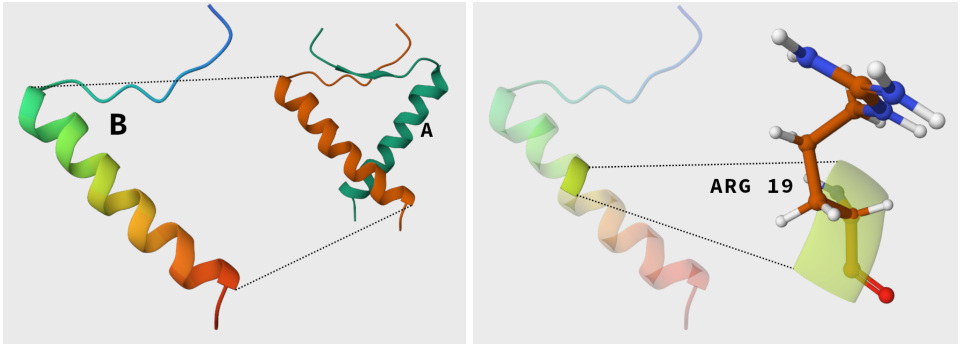
Atoms are the basic building blocks of matter. Atoms are composed of protons, neutrons, and electrons. The number of protons determines the element, while the number of neutrons determines the isotope. The number of electrons determines the charge of the atom. Atoms are the smallest unit of matter that can take part in a chemical reaction.

Bonds are the connections between atoms that hold molecules together. There are different types of bonds, e.g. covalent bonds, ionic bonds, and hydrogen bonds.

A residue refers to a specific building block that remains after a chemical modification or enzymatic reaction. In proteins, residues refer to amino acids, which are connected through peptide bonds to form polypeptide chains. Residues are crucial in biochemistry because they determine the structure and function of biological molecules. The sequence of residues in a protein or nucleic acid, for instance, determines its three-dimensional structure and ultimately its biological activity.

Polymer chains (chains) are sequences of residues that are linked together. In the context of biomolecules, chains can be either polypeptide chains in proteins or polynucleotide chains in nucleic acids. The sequence and structure of these chains are crucial for understanding the function and properties of the biomolecules.

Alternative conformations, also referred to as conformational isomerism, are different conformations of the same molecule. They can be used to represent different states of a molecule, such as different protonation states or different ligand binding modes. Alternative conformations are important for understanding the structure and function of biomolecules, as they provide insight into how they interact with other molecules.



(a) Residues of chain B. (b) Atoms of residue ARG 19.

Figure 1.1: Hierarchy of structure 1A1U.

1.2 Partial atomic charges

TODO: this section will describe partial atomic charges, their applications, then it will describe how they are calculated using quantum mechanics, explain the need for empirical methods, describe empirical methods, and lastly mention chargefw2 and alphacharges as applications that use the empirical methods

Partial atomic charges are numerical values assigned to individual atoms within a molecule, representing their contribution to the overall charge distribution. These charges provide information about the electron density and the distribution of positive and negative charges within the molecule.

There are several applications for partial atomic charges in computational chemistry, including molecular dynamics simulations, molecular docking, and molecular mechanics.

- Molecular dynamics - are used to simulate the motion of molecules over time. It is used to study the dynamics of molecules, such as proteins and nucleic acids.

- Molecular docking is used to predict the binding affinity of a ligand to its receptor. It is used to study the interactions between molecules, such as proteins and nucleic acids.

- Molecular mechanics is used to calculate the energy of a molecule. It is used to study the stability of molecules, such as proteins and nucleic acids.

1.2.1 Empirical methods

However, the quantum mechanics of molecules is too complex to be solved analytically. Therefore, empirical methods are used to approximate the electron density and the distribution of positive and negative charges within the molecule.

1.3 Chemical file formats

TODO: this section describes the file formats used during the thesis work; i will briefly describe SDF, MOL2, and PDB focus in more detail on the mmCIF file format: i will explain the data dictionaries; show one example molecule in each file format for SDF-MOL2 and PDB-mmCIF

Chemical file formats are used to store molecular data in a computer-readable format. There are many file formats for storing molecular data, each suited to a specific purpose. This section provides an overview of the most common file formats used in computational chemistry.

1.3.1 SDF

Structure-data file (SDF) is a widely used chemical file format for representing molecular structures and their associated properties. It is a text-based format that describes the atoms, bonds, and atomic coordinates of a molecule.

1.3.2 MOL2

The Mol2 file format is another text-based format for storing molecular structures and their associated properties. It can store multiple conformations of a molecule and is commonly used in molecular modeling and cheminformatics applications. The Mol2 format provides more

flexibility and additional features compared to the SDF format, such as support for multiple substructures and atom types.

1.3.3 PDB

[1]

The Protein Data Bank (PDB) file format is a widely used format for storing three-dimensional structures of proteins, nucleic acids, and other macromolecules. PDB files contain information about the atomic coordinates, secondary structure, and other important details required for understanding macromolecular structures. The PDB format has been widely adopted in structural biology, bioinformatics, and related fields.

1.3.4 mmCIF

TODO: explain the need for creating the mmCIF file format, why isn't the PDB format enough (limited amount of rows, poor support for extending the format to add custom data)? describe the mmcif file format; make sure to explain the core data dictionaries, can mention binary cif

[1]

The macromolecular Crystallographic Information File (mmCIF) format is an extension of the CIF format, specifically designed for macromolecular structures. It is a text-based format that provides a more comprehensive and flexible representation of macromolecular crystallography data compared to the PDB format. One of the most important features of the mmCIF format is its support for data dictionaries. This allows users to define new data items and integrate additional information. In contrast to other formats, the mmCIF format does not impose limits on column width and entry count, making it more flexible and accommodating for storing large amounts of data.

TODO: add example image + better explanation

1.4 Color interpolation

TODO: this section will briefly describe color interpolation, add math equation so that it looks pro, add example of interpolations used in the partial charges color theme

Color interpolation is the process of creating new colors by mixing two or more colors together. It is a common technique used in computer graphics and digital image processing to create smooth transitions between colors.

Color interpolation works by calculating the intermediate colors between two or more given colors. This is typically done by taking a weighted average of the red, green, and blue values of the colors being interpolated.

TODO: add math equation + image of red,white and white,blue interpolation

2 Visualizing molecular data

2.1 Types of visualizations

TODO: thinking about merging the types of visualizations into one section like i did with the molecular structure

There are several methods to represent molecular data, each serving a different purpose and providing a different level of detail. The methods most relevant to this work are the following three types: ball and stick, surface, and cartoon.

2.1.1 Ball and stick

The ball and stick model represents atoms as spheres and bonds as cylindrical connections between these spheres. This model provides a simple and intuitive visualization of a molecule's atomic structure. It highlights individual atoms and their bonds, including their bond types. However, it may not accurately represent the spatial relationships between atoms in larger molecules or macromolecular complexes.

An example of a ball and stick model is shown in Figure 3.5(a).

2.1.2 Surface

Surface representations depict the three-dimensional shape of a molecule by displaying its solvent-accessible surface. This model provides a more accurate representation of the molecule's overall shape and size, making it especially useful for studying macromolecular interactions and the binding of small molecules. For example, surface visualization can be used to identify potential binding sites on a protein surface, which can then be targeted by drug molecules.

An example of a surface model is shown in Figure 3.5(b).

2.1.3 Cartoon

Cartoon representations simplify the molecular structure by focusing on the secondary structure elements of proteins and nucleic acids,

such as alpha helices, beta sheets, and loops. Alpha helices are often depicted as a spiral-like structures, whereas beta sheets as arrows. This type of visualization is particularly useful for visualizing large macromolecular complexes, as it highlights the overall organization and topology of the molecule without the clutter of atomic details. The simplification of the structure also makes it easier to understand the folding and dynamics of the molecule.

An example of a cartoon model is shown in Figure 3.5(c).

2.2 Coloring of molecular visualizations

TODO: select a handful of color themes; don't list them out; instead describe them in text and give examples of how they are used by researchers

Equally important as the type of visualization is the coloring scheme used to represent the molecule. Coloring is an essential aspect of molecular visualization, as it can provide additional information and help to emphasize specific features or properties of the molecule. Some common coloring schemes include:

- **Element symbol**: atoms are colored according to their chemical element (e.g., carbon in grey, oxygen in red, nitrogen in blue).
- **Partial charge**: atoms are colored according to their partial charge (e.g., positive in blue, negative in red).
- **B-factor**: atoms are colored according to their B-factor value (e.g., low in blue, high in red).
- **Residue index**: atoms or residues are colored according to their residue index (e.g., low in blue, high in red).
- **Residue type**: atoms or residues are colored according to their residue type (e.g., hydrophobic in green, hydrophilic in blue).
- **Chain identifier**: atoms or residues are colored according to their chain identifier (e.g., chain A in blue, chain B in red).

- **Atom type**: atoms are colored according to their atom type (e.g., carbon in grey, oxygen in red, nitrogen in blue).

- **Secondary structure**: Residues are colored according to their secondary structure type (e.g., alpha helix in red, beta sheet in blue).

- **Hydrophobicity**: atoms or residues are colored according to their hydrophobicity. Hydrophobic residues are depicted in green, hydrophilic residues in blue.

2.3 Visualization software

TODO: describe the web based

There are numerous software tools available for visualizing molecular data, with varying levels of complexity, customization, and features. Some of the most popular tools include MolScript [2], PyMOL [3], VMD [4], and ChimeraX [5]. More recently, web-based tools, such as NGL Viewer [6], LiteMol [7], and Mol* [8], have become increasingly popular due to their accessibility and ease of use. Furthermore, these web-based tools are especially useful for visualizing large macromolecular complexes, which are becoming increasingly common due to advances in structural biology techniques such as crystallography and electron microscopy. [8]

In the following sections, we will discuss the two web-based molecular visualization tools used in this work: LiteMol and Mol*.

2.3.1 Litemol

LiteMol is an open-source, web-native molecular visualization tool that supports various file formats and offers a user-friendly interface for creating visualizations. LiteMol provides essential visualization types, including ball and stick, surface, and cartoon representations, as well as options for customizing colors, lighting, and other display settings. The web-based nature of LiteMol makes it easily accessible and platform-independent.

The LiteMol suite is a freely available tool for visualizing large macromolecular structure datasets, which consists of three components: data delivery services, a compression format, and a lightweight 3D molecular viewer. It enables fast delivery and visualization of large datasets and is compatible with modern web browsers and mobile devices, making it accessible to users with and without structural biology expertise. The tool addresses the challenges of delivering and visualizing large structural data sets, which are becoming increasingly available due to advances in electron microscopy and other techniques. [7]

2.3.2 Mol*

Mol* (/'molstar/) is another web-native molecular visualization tool, developed as part of the wwPDB OneDep system for macromolecular structure deposition and validation. Mol* offers a wide range of visualization options, including advanced features such as electron density maps and validation reports. Mol* supports many file formats, including PDB, mmCIF, and PDBx/mmJSON. Like LiteMol, Mol* is platform-independent and can be accessed from any web browser. [8]

Mol* emphasizes interactivity and offers various tools for manipulating and analyzing the molecular structure, such as distance and angle measurements, selection and display of specific residues, and custom coloring schemes. Additionally, Mol* provides integration with external databases and services, such as UniProt, PDBe, and RCSB PDB, enabling users to quickly access related information and resources.

3 Mol* partial charges extension

Visualizing partial atomic charges in molecules is an essential aspect of computational chemistry research, aiding in analyzing complex molecular structures. Mol* provides an extensive range of features for users to explore molecular structures. However, the tool lacks the functionality to color and label atoms and residues based on their partial atomic charges. This can be a considerable limitation for researchers. In response to this need, we have created an extension to Mol* that addresses this limitation.

It should be noted that the Mol* viewer predecessor, Litemol, supported this functionality. However, since Litemol is no longer supported, we saw the need to bring this functionality to Mol*.

This chapter describes the requirements for the extension, the custom mmCIF categories necessary for storing the partial atomic charges, and the implementation of the extension itself.

3.1 Requirements

TODO: describe what residue charges are
TODO: describe what each representation should visualize when colored using partial charges coloring

Firstly, the extension should enable the coloring of atoms and residues based on their partial atomic charges. Secondly, it should describe the charge values of the atoms and residues. Thirdly, the extension should allow the user to provide multiple charge sets for a single structure and select which one to display. Finally, the extension should be seamlessly integrated into the Mol* library, facilitating access to its features and functionality.

3.2 Mol* state tree

After loading a structure the Mol* viewer loads the structure into a state tree. The state tree is a tree structure that contains node which can be anything really. At the leaves of the tree are 3D Representations

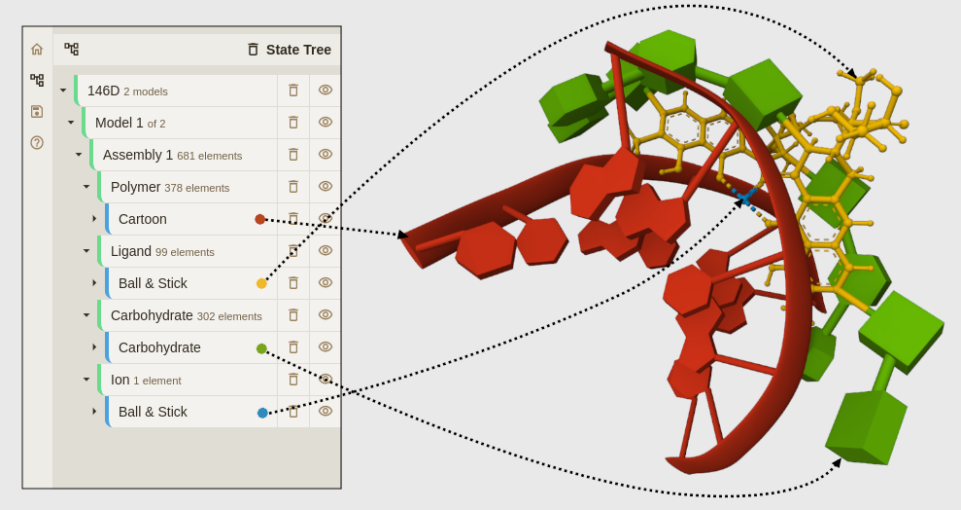


Figure 3.1: Mol* state tree.

of the parent selections. The selections are just a collection of atoms that should be visualized.

3.3 Custom mmCIF categories

To store partial atomic charges within a single file, we developed custom categories within the mmCIF format. The mmCIF format was chosen because it is widely used in the field of structural biology and offers several advantages over other formats, as discussed in 1.3.4. The custom categories allow us to store information about the partial atomic charges separately from the other structural data, while still being able to access it within the same file. Storing all data in one file was important as it allowed for easier management and distribution of the data. If the charges were stored separately, we would have to provide the charge data to Mol* in a different way e.g. through custom import controls.

We used two separate categories for this purpose: one to store the partial charge values for each atom in the structure, and another to store metadata about the charge sets.

The category `_sb_ncbr_partial_atomic_charges` maps together the atoms of the structure and their charges. The category has three attributes:

- `type_id` - pointer to the `_sb_ncbr_partial_atomic_charges_meta.id` item
- `atom_id` - pointer to the `_atom_site.id` item described in 1.3.4
- `charge` - partial charge value for the atom

The category `_sb_ncbr_partial_atomic_charges_meta` is dedicated to storing metadata about the charge sets. The metadata category has the following attributes:

- `id` - unique identifier for the charge set
- `type` - type of the calculation method (e.g. 'empirical', 'quantum')
- `method` - computation method used to calculate the charge set (e.g. 'EQeq', 'EEM/Racek 2016 (ccd2016_npa)')

Figure 3.2 provides a detailed illustration of the custom mmCIF categories and their relationships.

3.4 Implementation

This section will detail the implementation of the extension. The extension consists of multiple providers. Each provider serves a distinct functionality, such as supplying the partial charge data and coloring the structural elements based on their charges. The providers will be described in detail in the following subsections.

The extension was created using TypeScript, a superset of JavaScript that adds static typing and other features to the language. The Mol* library is also written in TypeScript, so the extension was written in the same language to ensure compatibility.

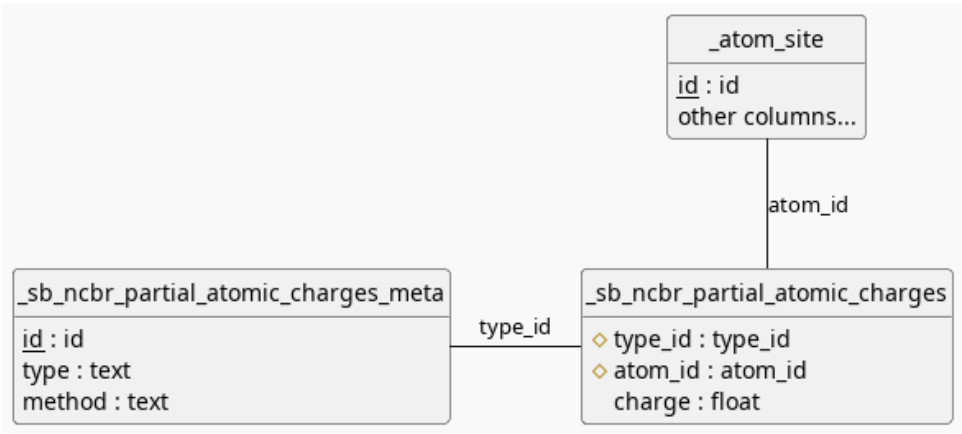


Figure 3.2: Diagram of custom mmCIF categories.

3.4.1 Property provider

In order to retrieve the charges from the mmCIF file, it is necessary to parse the file. This is done by the Mol* library, which parses the mmCIF file and provides the parsed mmCIF file data in the form of a `MmcifFormat` object. The purpose of this provider is to process the charge data from this object and supply the charge data to the rest of the extension providers through a custom property. The interface of this property is depicted in Figure **??**.

The atom charges are stored in the `typeIdToAtomIdToCharge` map. The map is indexed by the charge set (typeId) and the atom id. The atom id is a pointer to the atom_site.id. item in the mmCIF file. The atom charges are retrieved from the mmCIF file by iterating over the atom_site.id category and retrieving the charge values for each atom. The charge values are then stored in the `typeIdToAtomIdToCharge` map.

The residue charges are calculated by summing the charges of the atoms that make up the residue. The residue charge is then stored in the `typeIdToResidueIdToCharge` map.

The maximum absolute charge values of the atoms and residues are calculated and stored in the `maxAbsoluteAtomCharges` and `maxAbsoluteResidueCharges` maps. These maps are used in the color theme provider to normalize the charges to the range of 0 to 1. Additionally, the maximum

absolute charge values are used to calculate the color interpolations in the color theme provider. Additionally, the maximum absolute charge of both atoms and residues is calculated and stored in the `maxAbsoluteChargesAll` map.

Lastly, the method name used to calculate the charges of a given charge set is stored in the `typeIdToMethod` map. This map is used to display the method name in the UIs.

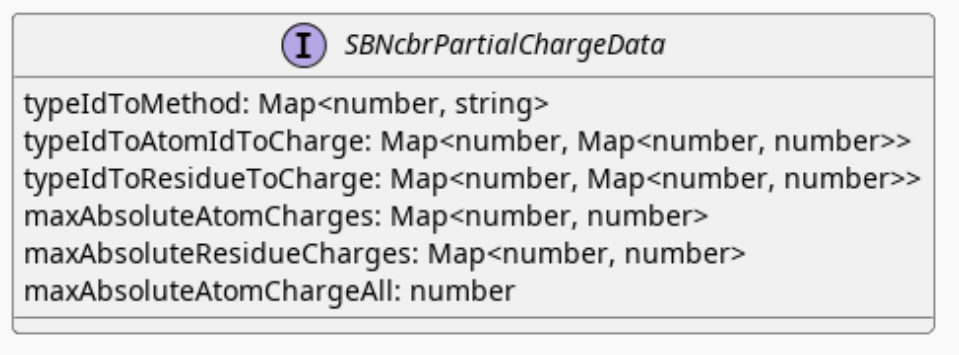


Figure 3.3: Interface of the custom model property for storing partial atomic charges.

3.4.2 Color theme provider

TODO: mention the color parameters

This provider serves as the central component of the extension, with its primary function being to assign colors to atoms and residues based on their charges. It achieves this by using the ColorTheme API provided by Mol*. The ColorTheme API is a mechanism for assigning colors to structural elements of a molecule. These structural elements can be atoms, residues, bonds, and so on. The API is based on the concept of a ColorTheme object, which is a collection of color assignments for structural elements. The ColorTheme object is then used by the Mol* library to color the structural elements of the molecule.

For the purposes of this extension, it was necessary to color two structural elements - atoms and residues. For both of these structural

```
type TypeId = number;
type IdToCharge = Map<number, number>;
export interface SBNcbrPartialChargeData {
    typeIdToMethod: Map<TypeId, string>;
    typeIdToAtomIdToCharge: Map<TypeId, IdToCharge>;
    typeIdToResidueToCharge: Map<TypeId, IdToCharge>;
    maxAbsoluteAtomCharges: IdToCharge;
    maxAbsoluteResidueCharges: IdToCharge;
    maxAbsoluteAtomChargeAll: number;
    params: PartialChargesPropertyParams;
}
```

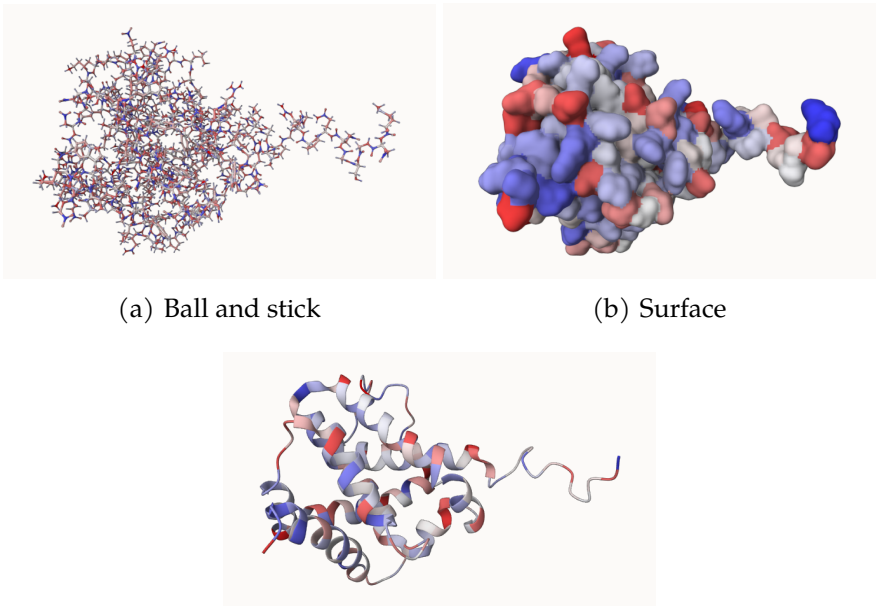
Figure 3.4: Interface of the custom model property for storing partial atomic charges.

elements the charges were retrieved from the provider described in the previous section **??**, which provided charges for atoms and residues.

To establish the color for a given charge, two color interpolations are employed: one for negative charges and another for positive charges. Atoms with positive charges receive a color from a white-to-blue color interpolation, while atoms with negative charges are assigned a color from a white-to-red color interpolation. These color interpolations are highlighted in Figure **??**.

3.4.3 Label provider

Having colored the structural elements, it was also necessary to create a label provider, which would assign labels that describe the charge of the structural element. In order to determine which element is highlighted, Mol* uses the object Loci. A Loci object is utilized for general selections and highlights. Consequently, it is essential to first extract the location from the Loci object in order to obtain the atom ID. The charge is acquired from the property provider, and the label is an HTML string that conveys the charge of the atom or residue. An example of the label can be seen in in the right-hand corner in figure **??**.



(a) Ball and stick

(b) Surface

(c) Cartoon

Figure 3.5: Partial charges color theme for different visualization types.

3.4.4 Controls

The controls are implemented automatically by the Mol* library based on the parameters of the providers. The user has access to controls of the charge set and the color theme. The charge set controls allow the user to select the charge set to display. The color theme controls allow the user to specify the following parameters:

- **Charge Range**: Sets the range of the color interpolation
- **Use Range**: Toggles whether the range of the color interpolation is automatically calculated or manually specified.
- **Charge Type**: Selects whether to display the partial atomic charges or the partial residue charges.

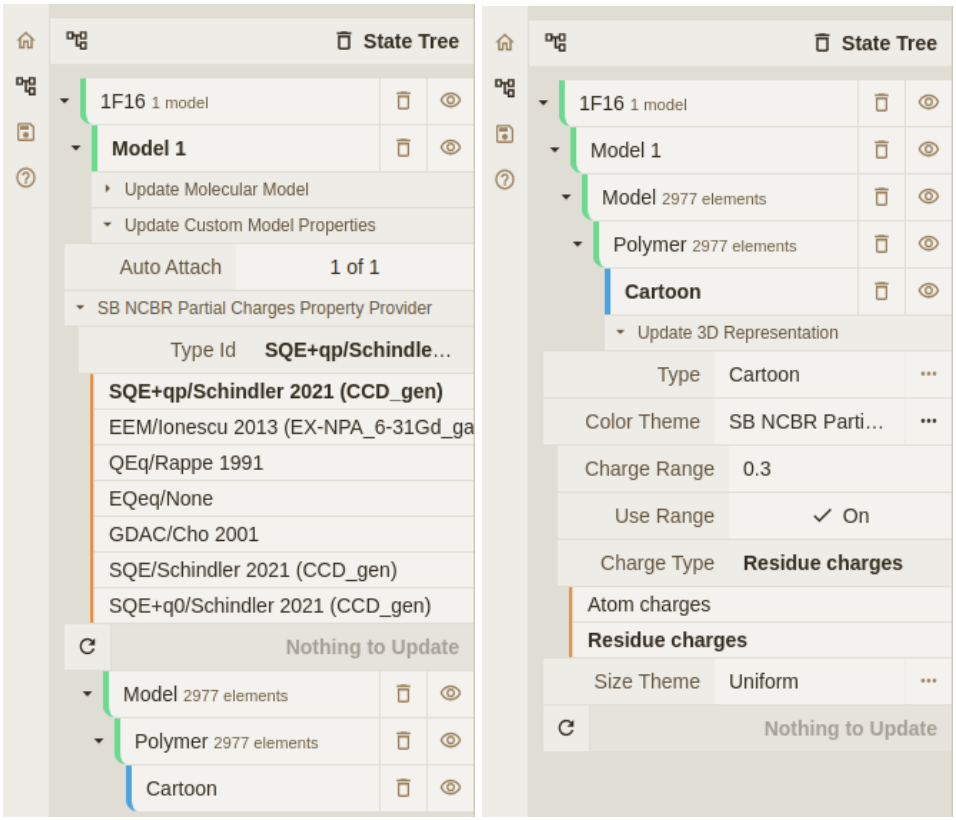
The controls are depicted in Figure 3.6.

3.5 Mol* viewer plugin

In addition to the partial charges extension for the Mol* library, it was necessary to create a custom Mol* viewer plugin. By using a plugin, it is possible to create custom behavior, which is not provided by the standard Mol* viewer.

The plugin is designed as a Typescript class. The structure of the class is depicted in Figure 3.7. It contains a method to create the plugin and another to load a structure file. Furthermore, it includes four attributes, each an object with methods that act as an interface for managing the plugin:

- *Charges* - provides functions for setting the charge set and retrieving information about the charge sets.
- *Color* - provides functions for setting the color theme.
- *Type* - provides functions for setting the visualization type.
- *Behavior* - provides a function for focusing on a specific atom.



(a) Charge set controls (b) Color theme controls

Figure 3.6: Extension controls.

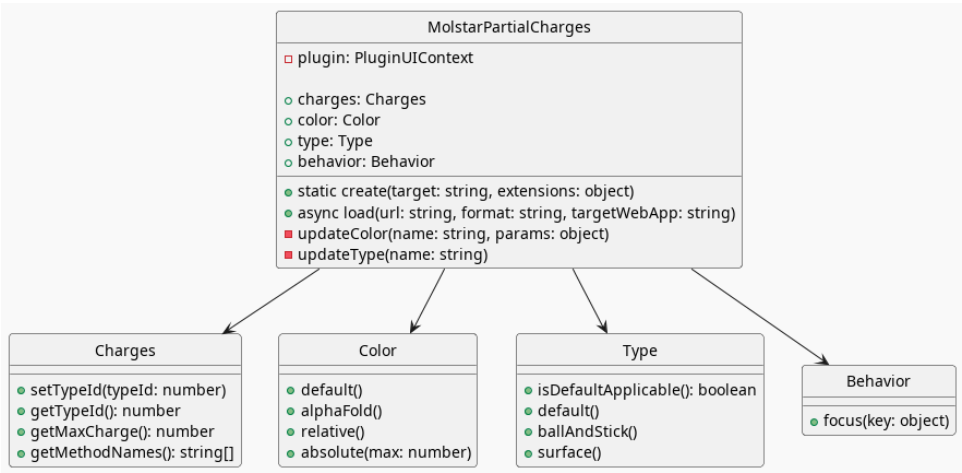


Figure 3.7: Diagram of the custom Mol* viewer class .

The *Color* and *Type* attributes set the color theme and visualization type through the private methods *updateColor* and *updateType*. These methods change the color theme and visualization type by iterating over the Mol* state tree and updating the representation nodes.

The focus functionality, implemented by the *Behavior* attribute, is achieved by first selecting the desired atom using a query language. Mol* supports multiple query languages: MolScript, PyMOL, VMD, and Jmol. The query language used in this plugin is MolScript. The query language is used to uniquely identify and select the atom by the following attributes of the mmCIF category *atom_site*:

- *labelCompId* - identifier for the chain containing the residue with the atom.
- *labelSeqId* - sequence number of the residue containing the atom.
- *labelAtomId* - identifier for the atom in the residue.

The MolScript query selection is depicted in Figure 3.8. Once the atom is selected, the selection is highlighted and focused on with methods provided by the Mol* library.

```
  const selection = Script.getStructureSelection(
    (Q) =>
        Q.struct.generator.atomGroups({
            'atom-test': Q.core.logic.and([
                Q.core.rel.eq([Q.struct.atomProperty.macromolecular.
                label_comp_id(), labelCompId]),
                Q.core.rel.eq([Q.struct.atomProperty.macromolecular.
                label_seq_id(), labelSeqId]),
                Q.core.rel.eq([Q.struct.atomProperty.macromolecular.
                label_atom_id(), labelAtomId]),
            ]),
        }),
    data
);
```

Figure 3.8: MolScript query for selecting an atom.

3.5.1 Technologies used during development

The plugin was developed using Vite. Vite is a build tool that was created to address some of the problems faced by developers when building large applications with traditional build tools such as Webpack. One of the main advantages of Vite is that it provides a fast development server. The speed of the development server is achieved by pre-bundling the static project dependencies with esbuild, a modern bundler written in Go, and serving the project source code with native ES module imports. [**vite**]

The plugin is written in Typescript and uses the Mol* library. The plugin is bundled using Rollup [**rollup**], a module bundler for Javascript, and published to NPM, a package manager for Javascript applications. [**npm**]

4 Atomic Charge Calculator II

Atomic Charge Calculator II (ACC2) is a web application for calculating partial atomic charges for structure files. The application is built using Flask for the backend and Javascript with Bootstrap for the frontend. The core of the ACC2 application is the ChargeFW2 program, which is used to calculate the partial atomic charges. For visualizing the calculation results, the application uses the Litemol software. [9]

Since the Litemol software is no longer supported, it was necessary to replace it with its modern counterpart, Mol*. This chapter describes the incorporation of the Mol* viewer into the ACC2 application. We first discuss the modifications to the ChargeFW2 program, then explain the changes made to the backend and frontend to support calculations of multiple charge sets, and lastly we describe the integration of the Mol* viewer into the ACC2 application.

4.1 Extension of ChargeFW2

As described in 1.2.1, ChargeFW2 is a C++ program for computing partial atomic charges. The program supports the following input file formats: SDF, MOL2, PDB, and mmCIF, all of which are described in Section 1.3. The program parses the necessary atom and bond data for each molecule in the input file and stores it in a `Molecule` object. These objects are then stored in a `MoleculeSet` object over which the program then iterates and calculates the partial atomic charges for each molecule.

The program outputs the charge results in multiple file formats. The following table 4.1 lists the output file formats generated by ChargeFW2 for the corresponding input file formats.

Table 4.1: ChargeFW2 output file formats.

Input format	**Output formats**
SDF, MOL2	TXT, MOL2
PDB	TXT, PQR
mmCIF	TXT, PQR, mmCIF

As can be seen in Table 4.1, ChargeFW2 already outputs a mmCIF file. The format of this mmCIF file stores the partial atomic charges in a custom `_atom_site.fw2_charge` item. This format, however, is not compatible with the Mol* viewer, which expects the partial atomic charges to be stored in the custom mmCIF categories described in Section 3.3. Therefore, it was necessary to modify ChargeFW2 to output the partial atomic charges in the custom mmCIF categories.

The following subsections describe in detail how ChargeFW2 was modified to produce the custom mmCIF file format for all the input files, regardless of the input file format.

4.1.1 mmCIF

For mmCIF input files, the program simply appends the custom categories to the input file. To add the categories, the program uses the GEMMI library [10] to parse the input mmCIF file into a `Block` object, which provides access to all mmCIF categories in the file. Before appending the charge categories, the program first removes alternative conformations from the `Block` object. This is necessary because the Mol* viewer does not support alternative conformations. The program then appends the custom charge categories to the `Block` object and writes the `Block` object to a new mmCIF file.

4.1.2 PDB

The PDB input files are first converted to mmCIF format using the GEMMI library. To achieve this, the PDB file is first parsed into a `Structure` object, which is then converted into a `Block` object. After converting the PDB file to mmCIF, it was necessary to remove the `_chem_comp` category from the `Block` object. This category is generated by GEMMI by default, however, it is not necessary for the Mol* viewer and its presence causes the viewer to visualize the structure incorrectly. The program then proceeds in the same way as for the mmCIF input files.

4.1.3 SDF and MOL2

Both SDF and MOL2 input files are converted to mmCIF format using the atom and bond data stored in the `Molecule` object. The program first creates a `Block` object, which it populates with the `_atom_site` category using the atom data. Secondly, it adds the `_chem_comp_bond` category using the bond data. The later category describes the bonds between the atoms and is necessary for the Mol* viewer to visualize the bond orders correctly. The program then proceeds in the same way as for the mmCIF input files.

Result

For each calculation the ChargeFW2 produces a mmCIF file, which is compatible with the Mol* viewer. The Table 4.2 lists the updated output file formats generated by ChargeFW2 for the corresponding input file formats.

Table 4.2: Updated ChargeFW2 output file formats.

Input format	Output formats
SDF, MOL2	TXT, mmCIF, MOL2
PDB, mmCIF	TXT, mmCIF, PQR

4.2 Multicharge support

The ACC2 application allows the user to select the method and parameters that will be used to calculate the partial atomic charges for the user's input file. Once the user confirms their selection, the application sends it to the backend, which runs the ChargeFW2 program with the chosen method and parameters. The application then displays the results of the calculation to the user.

However, the application only supports one calculation per request and does not support the visualization of multiple charge sets. This limitation was previously caused by the Litemol viewer, which only supported one charge set per structure. Since the Mol* viewer supports multiple charge sets, the ACC2 application was modified to support multiple charge sets as well.

The following subsections describe the necessary changes made to the ACC2 application to facilitate multiple calculations per one request and the visualization of multiple charge sets.

4.2.1 Frontend changes

As explained in the previous section, the ACC2 application used to limit the user to selecting only one combination of method and parameters. This limitation was resolved by introducing new controls to the calculation setup.

Figure 4.1 shows the updated setup page for the ACC2 application. The page now contains a list of calculations, where each calculation represents a combination of method and parameters. The list is initially empty and the user can add a new calculation to the list by selecting the desired method and parameters from the drop-down menus and clicking the *Add to calculation* button. The user can also remove any calculation from the list by clicking the cross button next to the calculation name.

Once the user clicks the *Compute* button, the list of calculations is sent to the backend where it is used to generate the desired charge sets for the user's input file.

4.2.2 Backend changes

The Flask backend of the ACC2 application was modified to support multiple charge calculations per request. The backend now receives a list of calculations from the frontend, which it uses to generate the desired charge sets for the user's input file. It does this by running the ChargeFW2 program for each calculation in the list. For each calculation the ChargeFW2 generates the output file formats described in Table 4.2. The backend then parses the charge values from the output TXT files and stores them in a dictionary, creating a mapping between the charge set name and the charge values.

After all of the calculations are completed, the dictionary is used to generate a single mmCIF file. Firstly, the backend parses one of the output mmCIF files generated by ChargeFW2 into a `Block` object using the GEMMI library. The backend then iterates over the dictionary and appends the charge values to the `Block` object under the custom charge

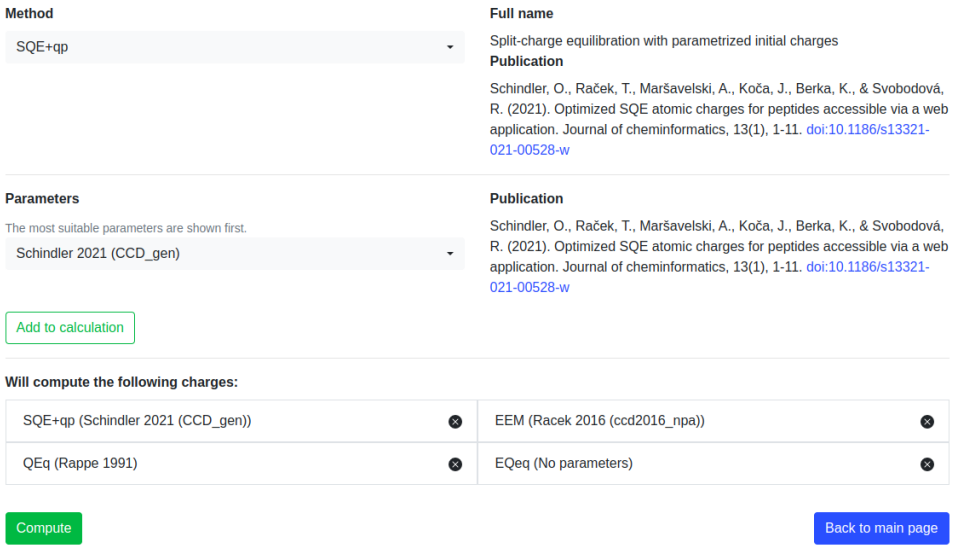


Figure 4.1: Updated setup page for the ACC2 application.

categories. Lastly, the backend writes the `Block` object to a new mmCIF file. This mmCIF file is then sent to the frontend, where it is loaded into the Mol* viewer.

4.3 Mol* viewer integration

Figure 4.2 shows the result page for the ACC2 application with the Mol* viewer. The result page initializes the custom viewer plugin and loads the mmCIF file generated by the backend. The loaded structure can then be visualized as a ball and stick model, as a surface model, or if applicable as a cartoon model. The page also provides controls for coloring the structure by partial atomic charges or by the element symbol of each atom. The user can also set the maximum charge value, which will be used for the partial atomic charges coloring.

The result page also contains two dropdown menus. The structure dropdown menu contains all the structures provided by the user in the input file. The charge set dropdown menu contains all the charge sets, which were generated by the backend according to the user's selection on the setup page.

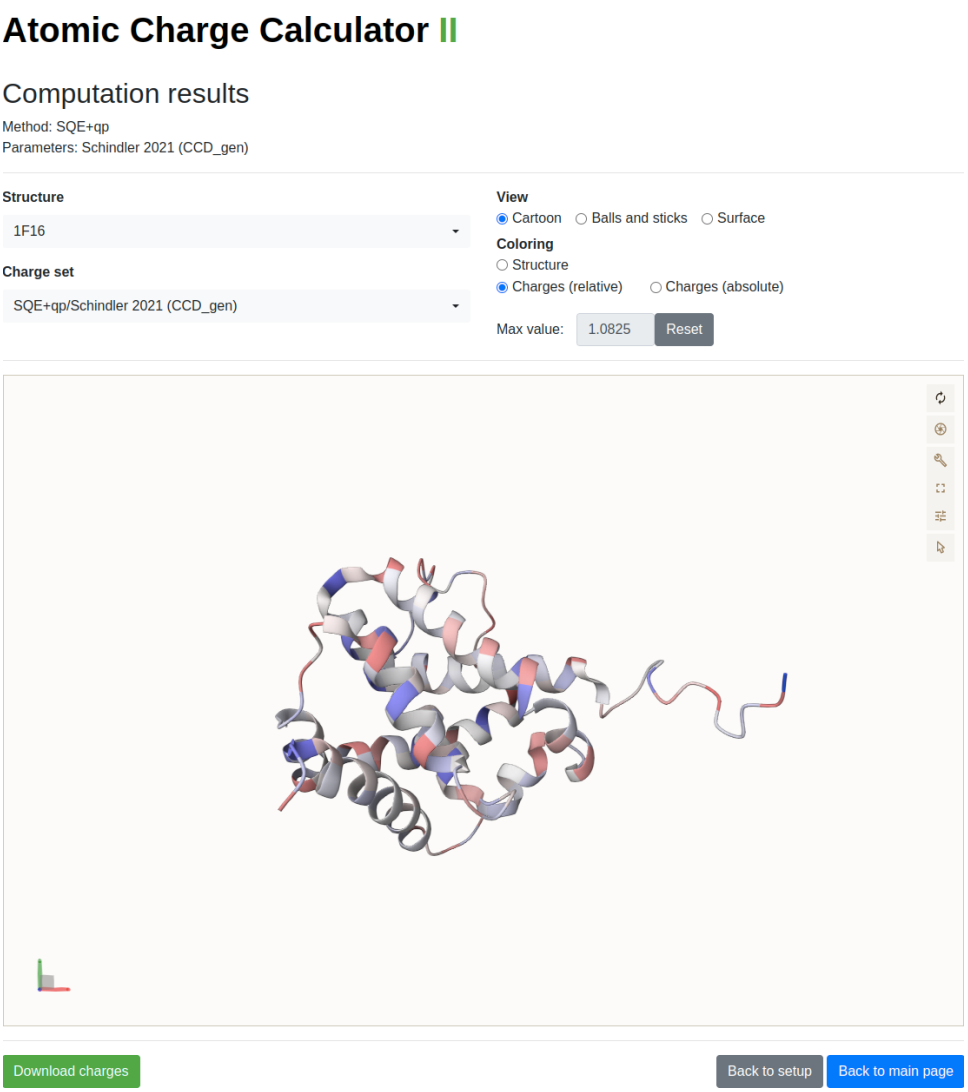


Figure 4.2: Updated result page for the ACC2 application.

Finally, the user can download all the output files generated by the backend in a ZIP archive by clicking the *Download charges* button. Figure 4.3 shows the directory structure of the ZIP archive. The `cif` directory contains mmCIF files for each structure with all the charge sets. The name of each mmCIF file is the same as the name of the corresponding structure. The names of each file in the `txt`, `mol2`, and `pqr` directories are prefixed with the name of the structure and the name of the charge set (i.e. `<structure>-<parameters>.<extension>`).

```
charges
├── cif
│   └── 1f16.fw2.cif
├── mol2
├── pqr
│   ├── 1f16-EEM_00_NEEMP_ccd2016_npa.pqr
│   └── 1f16-SQEqp_10_Schindler2021_CCD_gen.pqr
│
└── txt
    ├── 1f16-EEM_00_NEEMP_ccd2016_npa.txt
    └── 1f16-SQEqp_10_Schindler2021_CCD_gen.txt
```

Figure 4.3: Directory structure of the output ZIP archive.

5 AlphaCharges

AlphaCharges (αCharges) is a web application for calculating partial atomic charges for structures from AlphaFoldDB, a database of protein structures predicted by AlphaFold2 [11]. The AlphaCharges application protonates the protein structures (i.e. adds hydrogens) and calculates the partial atomic charges for the structures with the SQE+q0 empirical method [12]. [13]

During the development of the AlphaCharges application, the Mol* viewer was incorporated into the application to visualize the calculated partial atomic charges. The following sections describe the changes made to the Mol* viewer to support the AlphaCharges application and the integration of the Mol* viewer into the application.

5.1 Integration of the the Mol* viewer

TODO: explain the conversion of PDB into mmCIF and addition of ma_qa mmCIF categories with confidence scores

The AlphaCharges application shares the architecture with the ACC2 application. The frontend is written in Javascript with the Bootstrap framework, the backend is written in Python and uses the Flask framework. This shared architecture allowed the integration of the Mol* viewer into the AlphaCharges application with minimal adjustments. As with the ACC2 application, the Mol* viewer is incorporated into the result page by initializing the plugin viewer, loading the mmCIF file generated by the backend, and mounting the controls for managing the viewer.

The creators of the AlphaCharge application requested the addition of two new features to the Mol* viewer. Firstly, the creators requested the addition of a new color theme for the coloring of the structure by the pLDDT confidence score. Secondly, the creators requested a focus control for the viewer, which would allow the user to focus (i.e. zoom in and center) on a specific atom in the structure. The following subsections describe the implementation of the new features.

5.1.1 Coloring by pLDDT confidence score

AlphaFold2 generates a confidence score for each residue in the predicted protein structures using a measure called pLDDT. The confidence score is a value ranging between 0 and 100, where higher values indicate higher confidence levels. [14] The AlphaFoldDB includes the confidence scores along with the predicted protein structures in the mmCIF file format. These confidence scores are stored in the mmCIF file under the following categories:

TODO: the descriptions in the following list are taken from PDBx/mmCIF; how can i cite this?

- `_ma_qa_metric` - contains details of the metrics use to assess model quality.
- `_ma_qa_metric_local` - contains information about the local QA metrics calculated at the residue-level.
- `_ma_qa_metric_global` - contains information about the global QA metrics calculated at the model-level.

To support this feature, the Mol* viewer was extended to include a color theme provided by the Mol* library for visualizing the pLDDT confidence score. This pLDDT color theme uses a set of colors ranging from orange to blue, where orange indicates low confidence and blue indicates high confidence. The color theme is applied to the structure by mapping the pLDDT scores to the set of colors.

To incorporate this coloring in AlphaCharges, the custom Mol* viewer plugin was extended to include a function for switching to the pLDDT color theme. The AlphaCharges application then uses this function to add a new coloring control to the result page. This control is similar to the existing coloring controls, which allow the user to color the structure by the element symbol or by the partial atomic charges. The pLDDT coloring control is shown in Figure 5.1.

5.1.2 Focus on problematic atoms

The AlphaCharges application cannot calculate the partial atomic charges for all protein structures. Some structures can be incorrectly

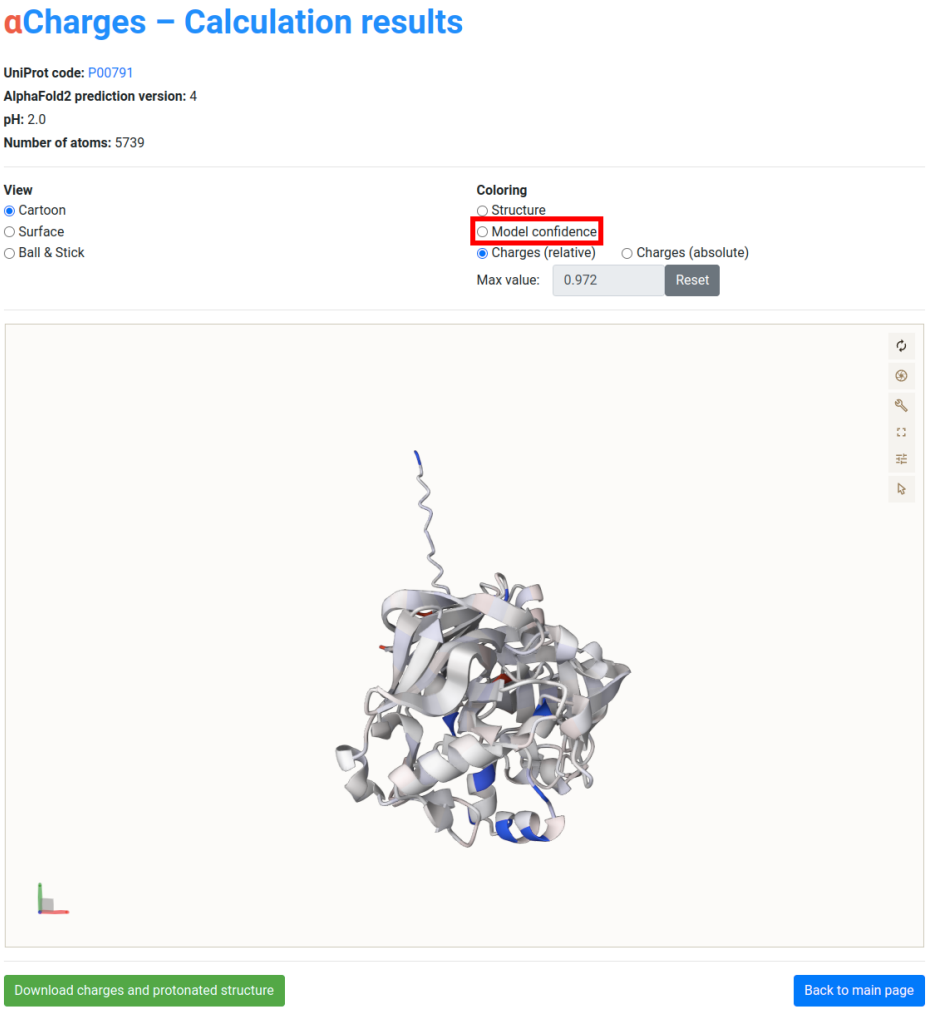


Figure 5.1: Result page of the AlphaCharges application. The control for switching to the pLDDT color theme is highlighted in red.

predicted by the AlphaFold2 algorithm or the application can fail to protonate the structure. [11] In such instances, the application automatically redirects the user to an error page, which displays the problematic structure and the error message.

We cover the implementation of the error page in more detail in the next section **??**. For the purpose of this section, it is sufficient to know that the error page contains an error message with the atoms that caused the error. The user has an option to click on the problematic atoms to focus on them in the structure. This feature needed to be implemented in the Mol* plugin viewer.

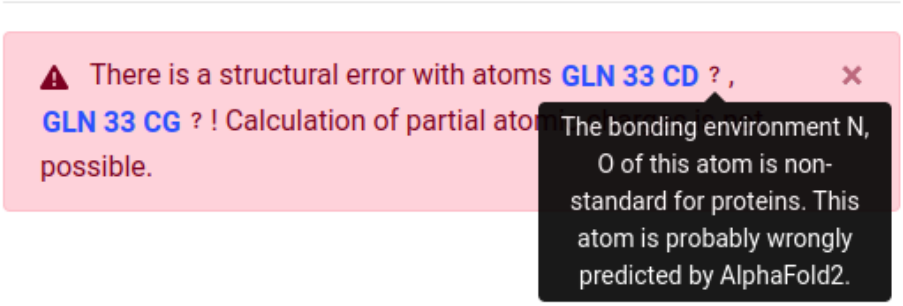


Figure 5.2: Explanation for error of atom *GLN 33 CD*.

The Mol* library provides a function to focus on a specific structure selection. The selection can be created programmatically using the Mol* query language. The query language is built using the Molscript language [2] and allows the user to select atoms based on their properties, such as their element symbol or their residue name.

To focus on the problematic atom *GLN 33 CD*, displayed in Figure 5.2, the query language was used to first select the atom with the residue name *GLN*, the residue number *33*, and the atom name *CD*. This combination of properties uniquely identifies the problematic atom in the structure. Then the selected atom was focused using the Mol* focus function. Figure 5.3 shows the Mol* viewer with the atom *GLN 33 CD* focused.

5.2 Problematic atoms webpage

TODO: should this be merged into the previous section?

The backend sends a JSON object to the frontend containing problematic atoms and the explanations. Figure **??** shows the structure of the JSON schema. This JSON object is then used by the frontend to display the problematic atoms and their explanations.

The problematic atoms are added to the error message as a link. When the user clicks on the link, a event listener will be trigger, which will make the Mol* viewer focus on the problematic atom. Figure 5.3 shows the Mol* viewer with the atom *GLN 33 CD* focused.

The user can also hover over a icon adjacent to the problematic atom to display the explanation for the probable cause of the error in a pop-up text. Figure 5.2 shows the explanation provided for the atom *GLN 33 CD*.

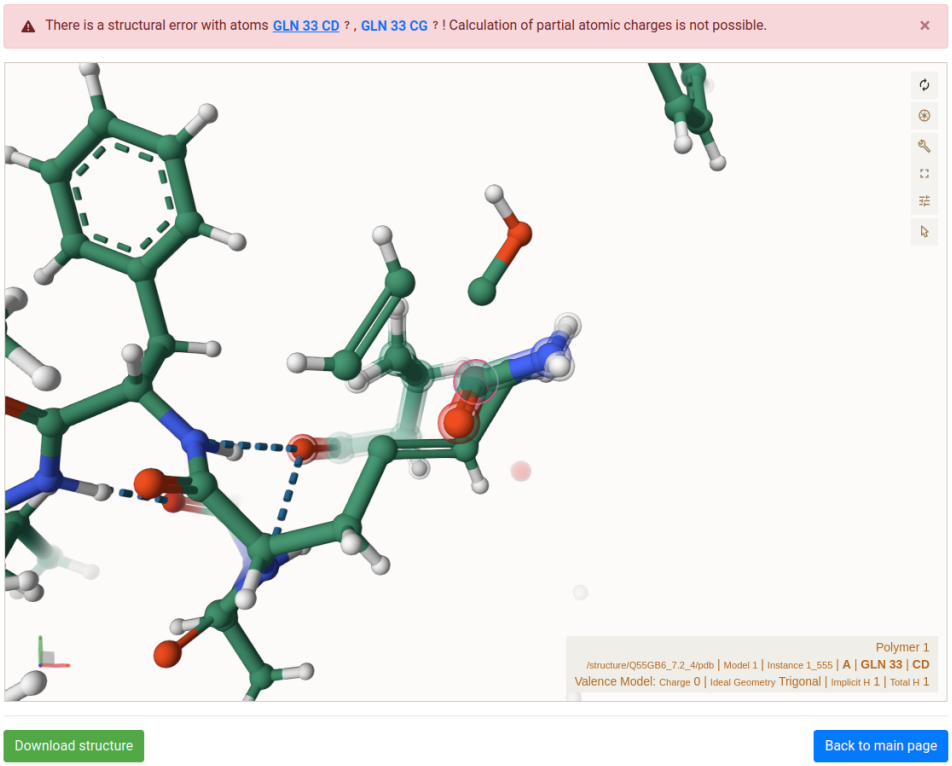


Figure 5.3: Focus on the atom GLN 33 CD, one of two problematic atoms in the structure with UniProt code Q55GB6.

Conclusion

Bibliography

1. GU, Jenny; BOURNE, Philip E. *Structural bioinformatics*. Vol. 44. John Wiley & Sons, 2009.
2. KRAULIS, P. J. *MOLSCRIPT*: a program to produce both detailed and schematic plots of protein structures. *Journal of Applied Crystallography*. 1991, vol. 24, no. 5, pp. 946–950. Available from DOI: 10.1107/S0021889891004399.
3. DELANO, Warren L et al. Pymol: An open-source molecular graphics tool. *CCP4 Newsl. Protein Crystallogr*. 2002, vol. 40, no. 1, pp. 82–92.
4. HUMPHREY, William; DALKE, Andrew; SCHULTEN, Klaus. VMD: Visual molecular dynamics. *Journal of Molecular Graphics*. 1996, vol. 14, no. 1, pp. 33–38. ISSN 0263-7855. Available from DOI: https://doi.org/10.1016/0263-7855(96)00018-5.
5. GODDARD, Thomas D.; HUANG, Conrad C.; MENG, Elaine C.; PETTERSEN, Eric F.; COUCH, Gregory S.; MORRIS, John H.; FERRIN, Thomas E. UCSF ChimeraX: Meeting modern challenges in visualization and analysis. *Protein Science*. 2018, vol. 27, no. 1, pp. 14–25. Available from DOI: https://doi.org/10.1002/pro.3235.
6. ROSE, Alexander S.; HILDEBRAND, Peter W. NGL Viewer: a web application for molecular visualization. *Nucleic Acids Research*. 2015, vol. 43, no. W1, W576–W579. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkv402.
7. SEHNAL, David; DESHPANDE, Mandar; VAŘEKOVÁ, Radka Svobodová; MIR, Saqib; BERKA, Karel; MIDLIK, Adam; PRAVDA, Lukáš; VELANKAR, Sameer; KOČA, Jaroslav. LiteMol suite: interactive web-based visualization of large-scale macromolecular structure data. *Nature Methods*. 2017, vol. 14, no. 12, pp. 1121–1122. ISSN 1548-7105. Available from DOI: 10.1038/nmeth.4499.
8. SEHNAL, David; BITTRICH, Sebastian; DESHPANDE, Mandar; SVOBODOVÁ, Radka; BERKA, Karel; BAZGIER, Václav; VELANKAR, Sameer; BURLEY, Stephen K; KOČA, Jaroslav; ROSE,

Alexander S. Mol* Viewer: modern web app for 3D visualization and analysis of large biomolecular structures. *Nucleic Acids Research*. 2021, vol. 49, no. W1, W431–W437. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkab314.

9. RAČEK, Tomáš; SCHINDLER, Ondřej; TOUŠEK, Dominik; HORSKÝ, Vladimír; BERKA, Karel; KOČA, Jaroslav; SVOBODOVÁ, Radka. Atomic Charge Calculator II: web-based tool for the calculation of partial atomic charges. *Nucleic Acids Research*. 2020, vol. 48, no. W1, W591–W596. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkaa367.

10. WOJDYR, Marcin. GEMMI: A library for structural biology. *Journal of Open Source Software*. 2022, vol. 7, no. 73, p. 4200. Available from DOI: 10.21105/joss.04200.

11. JUMPER, John; EVANS, Richard; PRITZEL, Alexander; GREEN, Tim; FIGURNOV, Michael; RONNEBERGER, Olaf; TUNYASUVUNAKOOL, Kathryn; BATES, Russ; ŽÍDEK, Augustin; POTAPENKO, Anna; BRIDGLAND, Alex; MEYER, Clemens; KOHL, Simon A. A.; BALLARD, Andrew J.; COWIE, Andrew; ROMERA-PAREDES, Bernardino; NIKOLOV, Stanislav; JAIN, Rishub; ADLER, Jonas; BACK, Trevor; PETERSEN, Stig; REIMAN, David; CLANCY, Ellen; ZIELINSKI, Michal; STEINEGGER, Martin; PACHOLSKA, Michalina; BERGHAMMER, Tamas; BODENSTEIN, Sebastian; SILVER, David; VINYALS, Oriol; SENIOR, Andrew W.; KAVUKCUOGLU, Koray; KOHLI, Pushmeet; HASSABIS, Demis. Highly accurate protein structure prediction with AlphaFold. *Nature*. 2021, vol. 596, no. 7873, pp. 583–589. ISSN 1476-4687. Available from DOI: 10.1038/s41586-021-03819-2.

12. SCHINDLER, Ondřej; RAČEK, Tomáš; MARŠAVELSKI, Aleksandra; KOČA, Jaroslav; BERKA, Karel; SVOBODOVÁ, Radka. Optimized SQE atomic charges for peptides accessible via a web application. *Journal of Cheminformatics*. 2021, vol. 13, no. 1, p. 45. ISSN 1758-2946. Available from DOI: 10.1186/s13321-021-00528-w.

13. SCHINDLER, Ondřej; BERKA, Karel; CANTARA, Alessio; KŘENEK, Aleš; TICHÝ, Dominik; RAČEK, Tomáš; SVOBODOVÁ, Radka. αCharges: partial atomic charges for AlphaFold structures in

high quality. *Nucleic Acids Research*. 2023. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkad349. gkad349.

14. VARADI, Mihaly; ANYANGO, Stephen; DESHPANDE, Mandar; NAIR, Sreenath; NATASSIA, Cindy; YORDANOVA, Galabina; YUAN, David; STROE, Oana; WOOD, Gemma; LAYDON, Agata; ŽÍDEK, Augustin; GREEN, Tim; TUNYASUVUNAKOOL, Kathryn; PETERSEN, Stig; JUMPER, John; CLANCY, Ellen; GREEN, Richard; VORA, Ankur; LUTFI, Mira; FIGURNOV, Michael; COWIE, Andrew; HOBBS, Nicole; KOHLI, Pushmeet; KLEYWEGT, Gerard; BIRNEY, Ewan; HASSABIS, Demis; VELANKAR, Sameer. AlphaFold Protein Structure Database: massively expanding the structural coverage of protein-sequence space with high-accuracy models. *Nucleic Acids Research*. 2021, vol. 50, no. D1, pp. D439–D444. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkab1061.