UNIVERSITY OF TARTU

FACULTY OF SCIENCE AND TECHNOLOGY

INSTITUTE OF PHYSICS

Material Science and Technology Curriculum

DANIEL AGBONLUAI IJEGBAI

An Open-Access Online Database of Core Electron Binding Energies from Gas Phase Photoelectron Spectroscopy

Master Thesis (30 EAP)

Supervisor: Juhan Matthias Kahk

TARTU, 2023

Fact sheet

Abstract

An Open-Access Online Database of Core Electron Binding Energies from Gas Phase Photoelectron Spectroscopy

Gas phase X-ray Photoelectron Spectroscopy (XPS) is one of the most widely used experimental techniques in the field of molecular physics. The central concept of core level XPS is the core electron binding energy – the energy required to remove a particular core electron from a given atom. Previously published core electron binding energies are frequently used to guide the interpretation of new experimental spectra. For XPS measurements of solid samples, there are large published databases of core electron binding energies that permit easy access to important reference data. In contrast, for gas phase XPS, no such databases have been previously available.

In this project, a new database of core electron binding energies from gas phase XPS is built. At launch, the database contains over 2,807 binding energy entries. For each entry, information about the molecule (chemical formula, molecular formula, name, InChI and SMILES identifiers), the reported binding energy (core level, binding energy in eV), and the literature reference for the original data are provided. The database can be accessed via an HTML interface hosted at xps.ut.ee, and the raw data can be downloaded from github.com/UT-XPS/gas-phase-database. The new database will be a valuable resource for future studies in the field of experimental molecular physics.

Keywords

X-ray Photoelectron Spectroscopy, XPS, gas phase XPS, molecular physics, core electron binding energy, database, cheminformatics

CERCS codes

P230 – Atomic and molecular physics, P300 – Analytical chemistry, P175 – Informatics, systems theory

Infoleht

Resümee

Avatud ligipääsuga veebipõhine gaasfaasi fotoelektronspektroskoopias mõõdetud siseelektronide seoseenergiate andmebaas

Gaasfaasi Röntgenfotoelektronspektroskoopia (XPS) on üks enimkasutatud eksperimentaalseid meetodeid molekulaarfüüsika uuringutes. Sisekihtide fotoelektronspektroskoopias on olulisim mõõdetav suurus siseelektroni seoseenergia – energia, mis tuleb kulutada ühe siseelektroni eemaldamiseks konkreetse aatomi juurest. Varasemalt publitseeritud siseelektronide seoseenergiaid kasutatakse tihti uute mõõtetulemuste tõlgendamisel. Seoseenergiate andmebaasid võimaldavad teadlastele kiiret ligipääsu olulistele referentsandmetele. Varasemalt on sellistesse andmebaasidesse koondatud tahketelt uurimisobjektidelt mõõdetud seoseenergiaid, ent mitte gaasfaasis mõõdetud seoseenergiaid.

Selle projekti käigus koostatakse uus gaasfaasis mõõdetud siseelektronide seoseenergiate andmebaas. Andmebaasi esialgne versioon sisaldab 2807 sissekannet. Iga sissekande juures on ära märgitud molekuli andmed (keemiline valem, molekulaarne valem, nimetus, InChI ja SMILES koodid), mõõdetud seoseenergia andmed (sisekiht, seoseenergia väärtus elektronvoltides), ja viide algandmetele. Andmetele ligi pääsemiseks on loodud HTML leht mis asub aadressil xps.ut.ee, kogu andmebaasi saab ka alla laadida lehelt github.com/UT-XPS/gas-phase-database. Koostatud andmebaasi eesmärk on olla väärtuslik tööriist uute uuringute tarbeks molekulaarse füüsika vallas.

Märksõnad

Röntgenfotoelektronspektroskoopia, XPS, gaasfaasi XPS, molekulaarne füüsika, siseelektroni seoseenergia, andmebaas, keminformaatika

CERCS koodid

P230 – Aatomi- ja molekulaarfüüsika, P300 – Analüütiline keemia, P175 – Informaatika, süsteemiteooria

Table of Contents

List of abbreviations

PES – photoelectron spectroscopy

XPS – x-ray photoelectron spectroscopy

BE – binding energy

CEBEs – core electron binding energies

IUPAC – International Union of Pure and Applied Chemistry

SMILES – Simplified molecular-input line-entry system

InChI - International Chemical Identifier

OPSIN - Open Parser for Structural IUPAC Nomenclature

HTML - Hypertext Markup Language

DOI - digital object identifier

py – python

h - planck's constant

ν - the frequency

eV – electron volts

NIST - National Institute of Standards and Technology

VB – valence band

PDF - Portable Document Format

CGI – common gateway interface

UPS - ultraviolet photoelectron spectroscopy

UV – ultraviolet

PEPICO - Photoelectron Photoion Coincidence

K_{max} – maximum kinetic energy

hν – photon energy

E_k - kinetic energy

ϕ_s - sample work function

ϕ_a - analyser work function

E_B – binding energy

Delta-SCF – delta self-consistent field

FinEstBeAMS - Finnish-Estonian beamline for Atmospheric and Materials Science

API – application programming interface

3D – three dimension

mbar – millibar

DOI - Digital Object Identifier

OCR - optical character recognition

UI – user interface

png – portable network graphics

Acknowledgements

A very huge thank you to my family with whom have provided me with the support to write this work. I would like to appreciate Dr. Juhan Matthias Kahk for giving me the opportunity to learn under his tutelage and his outstanding contributions during every phase of the project. My gratitude would not be complete without special mentions to Ihonosethale Oseghale and Precious Ejeh for their relentless efforts during the structuring of the database; their support cannot be overestimated in completing this work. Finally, huge thanks to my program coordinator, Hele Simon who is ever willing to listen and help and a special mention to Aivar Pere for his encouraging words always.

1. Introduction

Photoelectron spectroscopy (PES) is an experimental analytical technique that can be used to study the chemical composition and/or the electronic structure of a material or a molecule[1]. In core level X-ray photoelectron spectroscopy (XPS), the key quantity that is measured is the core electron binding energy[2], i.e. the energy required to remove a particular core electron from a particular atom. Core electron binding energies depend on the atomic number of the atom, and the core orbital in question (e.g. C 1s, O 1s, Cl $2p_{3/2}$, …), and also on the chemical environment of the atom[3],[4]. Therefore, a core level photoelectron spectrum contains information about the elemental and chemical composition of the sample[3].

In gas phase photoelectron spectroscopy, the sample is in the gaseous state. As such, the measured spectrum can be approximated as the sum of the spectra of the individual molecular species that are present in the analysis chamber[5]. Gas phase photoelectron spectroscopy is one of the most widely used experimental techniques in molecular physics.

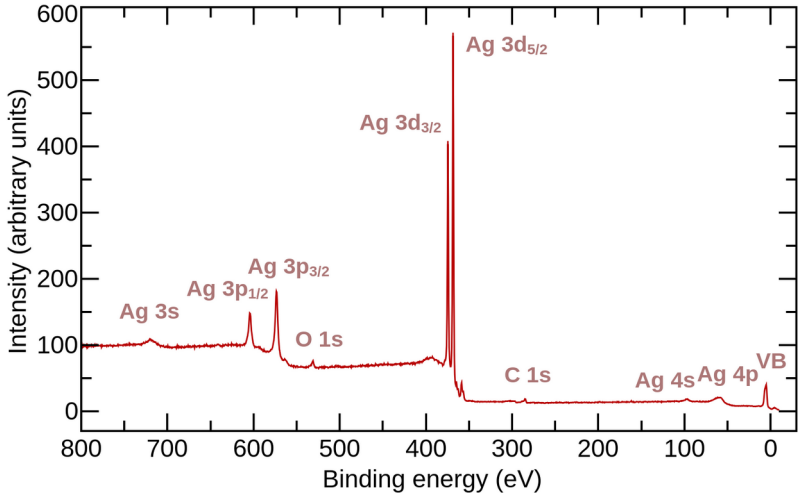


Figure 1: Survey spectrum of a MnO_2-carbon nanotube nanocomposite, covered with sputter deposited Ag. Due to the high surface sensitivity of XPS, the spectrum is dominated by the contributions from Ag metal. VB stands for valence band. Measured with Mg K-alpha radiation, hν = 1253.6 eV. This spectrum was provided by A. Kikas, Institute of Physics, University of Tartu.

An image of a typical core level photoelectron spectrum is shown in Figure 1. The spectrum can be divided into a slowly varying background, and one or several distinct “peaks”. Each peak corresponds to a transition to a different final state of the system.

In general, in XPS the presence of multiple peaks of a similar energy, e.g. multiple peaks in the C 1s region, indicates that the sample contains the same element (e.g. C) in several different chemical environments. Peak assignment is the process of determining the chemical environments that give rise to the peaks in the measured spectrum[6].

In XPS, reference data for the same compound or a similar compound is very often used to guide peak assignment. When analysing a complex spectrum, having access to good quality reference data is essential to be able to extract useful chemical information from the experimental measurements. Reference data can be obtained from control experiments of simpler materials or molecules, but these can be tedious and time-consuming, so in practice, published reference data from the scientific literature is often used[7],[8].

Spectral databanks and binding energy databases play an important role in practical XPS studies[6]. Databases allow researchers to rapidly access the reference data that are available for the system of interest, without having to personally comb through a large number of original research articles. For example, the NIST XPS database contains more than 27000 entries in total[9]. Several manufacturers of XPS instrumentation also develop their own databases, typically containing a smaller amount of high quality data for a simple, well-defined systems[10].

1.1 Problem statement

The NIST database, and other currently available XPS databases do not contain data from gas phase measurements, as most XPS users are primarily interested in studies of solids and surfaces. In order to access gas phase reference data, researchers have to manually search the literature, or rely on very out-of-date published collections of gas phase data, that are available as scanned pdf documents, such as the compilation of core electron binding energies from gas phase XPS from 1984 by Jolly et al. [11].

In this master's project, published research data from gas phase photoelectron spectroscopy and the tools of cheminformatics are used to build a searchable, open-access online database of core electron binding energies from gas phase XPS. The compiled database will be a valuable resource for future studies in the field of experimental molecular physics.

2. Background

2.1 Photoelectron spectroscopy

2.1.1 Basic principles

Photoelectron spectroscopy is based on the photoelectric effect[12], originally discovered in 1887 by Heinrich Hertz[13]. The photoelectric effect is the phenomenon whereby electrons are emitted from a sample, when it is exposed to light of a sufficiently short wavelength[14].

The photoelectric effect was rationalized in 1905 by A. Einstein, who received the Nobel prize in 1921 for his work[15]. According to Einstein's formula, the maximum kinetic energy of the emitted photoelectrons, K_{max}, is given by $K_{max} = h\nu - \phi$, where h is the planck's constant, ν is the frequency of the incident radiation, and ϕ is the work function of the material[16]. The conceptual significance of Einstein's formula is that it treats the incident radiation as a flow of particles called photons[17]. $h\nu$ is the photon's energy: the absorption of a photon can only lead to the emission of an electron, if the photon energy is sufficiently high to overcome the material's work function[18].

In photoelectron spectroscopy, the kinetic energies of all the emitted photoelectrons are measured. Energy conservation dictates that when leaving the sample, the kinetic energies are given by

$$E_k = h\nu - \phi_s - E_B,$$

where E_B is the electron's binding energy, i.e. the additional energy, beyond the work function, that is required to remove that electron from the sample, E_k is the photoelectron's kinetic energy, $h\nu$ is the photon energy, and ϕ_s is the work function of the sample.

When travelling from the sample to the analyser, the electrons must travel across a potential difference that is equal to the contact potential between the sample and the analyser, given by the difference between the work functions: $\phi_s - \phi_a$. Therefore, at the point of measurement, the kinetic energies of the photoelectrons are given by

$E_k = h\nu - \phi_a - E_B$.

The photoelectron spectrum is a spectrum of electron binding energies in a material or a molecule. As such, the spectrum contains information about the sample's electronic structure[19],[20].

In the low binding energy region, between $E_B = 0$ eV and $E_B \sim 15$ eV, the spectrum contains information about the valence electrons in the system[21]. At higher binding energies, the spectra of the various core states appear. Core electrons retain their atomic-like character even in complex materials, and therefore, the core states appear at characteristic energies in the photoelectron spectrum[16]. For example, in spectra of solids, C 1s states typically appear between 280 and 295 eV, O 1s states appear between 525 and 540 eV, and so on. This allows core level photoelectron spectroscopy to be used for determining the elemental composition of the sample[3],[4].

To a lesser degree, core electron binding energies also depend on the local chemical environment of the atom[22]. This dependence gives rise to chemical shifts in core electron binding energies, that are typically of the order of a few tenths of an eV to a few eV. As such, core level XPS can also be used to learn about the chemical composition of the sample[3].

2.1.2 Atomic Physics and Core Level Photoemission

For a closed shell molecule or a solid with no unpaired electrons, the total spin of the system in its electronic ground state is zero. The removal of one core electron leaves the system in a final state with the spin quantum number $S = 1/2$. The possible values of the total angular momentum quantum number, J, in the final state are determined by the Clebsch-Gordan series: $J = L + S, L + S - 1, \ldots, |L-S|$. L is the orbital angular momentum quantum number that is 0 for *s*-orbitals, 1 for *p*-orbitals, 2 for *d*-orbitals, and 3 for *f*-orbitals.

Therefore, the possible values of J are:

s-orbitals: $J = L + S = 0 + 1/2 = 1/2 = |L - S|$

p-orbitals: $J = L + S = 1 + 1/2 = 3/2$, and $J = L + S - 1 = 1 + 1/2 - 1 = 1/2 = |L - S|$

d-orbitals: $J = L + S = 2 + 1/2 = 5/2$, and $J = L + S - 1 = 2 + 1/2 - 1 = 3/2 = |L - S|$

f-orbitals: $J = L + S = 3 + 1/2 = 7/2$, and $J = L + S - 1 = 3 + 1/2 - 1 = 5/2 = |L - S|$

For photoemission from an s-orbital, only one peak is observed: n$s_{1/2}$, often just labelled n*s*, where n is the principal quantum number. For photoemission from a *p*-, *d*-, or *f*-orbital, a doublet is observed: n$p_{3/2}$ and n$p_{1/2}$, n$d_{5/2}$ and n$d_{3/2}$, n$f_{7/2}$ and n$f_{5/2}$. The separation between the two components of the doublet is termed the spin-orbit splitting.

The intensity ratio of the two components of the doublet is determined by the multiplicities of the states, given by 2*J*+1. The ratios are:

4/2 = 2/1 for an n*p* doublet

6/4 = 3/2 for an n*d* doublet

8/6 = 4/3 for an n*f* doublet

For *p*-, *d*-, and *f*-orbitals, core electron binding energies are typically reported for the more intense component of the doublet, e.g. n$p_{3/2}$, n$d_{5/2}$, and n$f_{7/2}$. For some relatively light elements with a 2*p* core level, e.g. Al and Si, the separation between the 2$p_{3/2}$ and 2$p_{1/2}$ components of the doublet is so small, that it cannot be easily resolved. In those cases, simply a 2*p* core electron binding energy is reported, that corresponds to a weighted average of the two components of the 2*p* doublet[23].

2.1.3 Instrumentation

A photoelectron spectrometer consists of, at the minimum, the following components: the photon source, the electron kinetic energy analyser. the detector, the analysis chamber, the vacuum system, and the utilities for sample manipulation. A schematic diagram of a basic photoelectron spectrometer is shown in Figure 2.

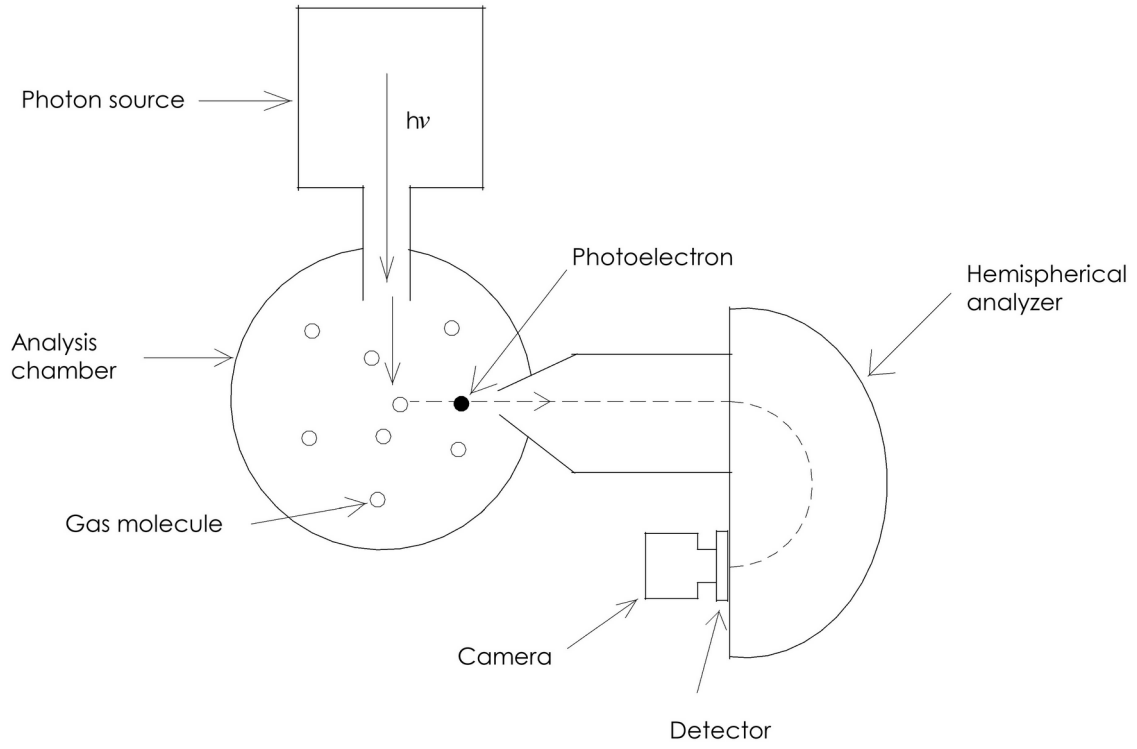


Figure 2: Schematic representation of a gas-phase photoelectron spectrometer

The photon source, usually combined with a monochromator, generates X-ray or UV radiation of a specific wavelength / photon energy. For instance, Al K-alpha radiation with a photon energy of 1487 eV is typically used in laboratory-based instruments. The incident radiation hits the sample in the analysis chamber, leading to the emission of photoelectrons. The emitted photoelectrons enter the kinetic energy analyser, that separates them according to their kinetic energies. After passing through the analyser, photoelectrons of a given kinetic energy hit the detector, and the rate at which electrons hit the detector screen is transmitted to a computer.

Photoelectron spectrometers must be operated at low pressures (high or ultrahigh vacuum), typically $< 10^{-6}$ mbar. There are several reasons for why low pressures are required. (i) High voltages are applied in a conventional X-ray anode and in the electron energy analyser. Therefore, dielectric breakdown (arcing) would occur at pressures above 10^{-5} mbar, making the measurement impossible, and causing damage to the instrument. (ii) At pressures that are excessively high, inelastic scattering of the emitted photoelectrons, as they travel through the gas atmosphere, would lead to a sharp drop in the detected signal. (iii) When photoelectron spectroscopy is used to study

clean surfaces, pressures < 10^{-10} mbar must be maintained in the analysis chamber, for otherwise the sample surface would be rapidly contaminated by gas molecules sticking to the surface[23].

2.1.4 Gas phase photoelectron spectroscopy

Photoelectron spectroscopy is most commonly used to characterize the surfaces of materials, but since the pioneering work of the Nobel Prize laureate Kai Siegbahn in the 1950s and 60s[24],[25] it has also been used to study gaseous samples. In gas phase XPS, the vacuum level is used as the point of reference[26], rather than the sample Fermi level, and therefore the equation for the binding energy simplifies to

$$E_B = h\nu - E_k$$

In gas phase measurements, the gaseous sample can be introduced into the analysis chamber in various ways, for instance via a leak valve, using an effusion cell, or using a molecular beam source[23]. Differential pumping can be used to permit relatively higher pressures in the analysis chamber, whilst keeping the photon source and the electron energy analyser under high or ultra high vacuum. Commonly used excitation sources include He discharge lamps for UV radiation, Al K-alpha or Mg K-alpha X-ray tubes, or synchrotron radiation. The gas phase end-station at the FinEstBeAMS beamline at the MAX IV synchrotron in Lund, Sweden is an example of a modern synchrotron based instrument for gas phase photoelectron spectroscopy.

Historically, gas phase photoelectron spectroscopy was used in studies of atomic physics, including a detailed investigation of the electronic structures of all noble gas elements by Siegbahn et al., and studies of the electronic structures of small molecules. More recently, gas phase photoemission has been used to study advanced aspects of molecular physics, such as the behaviour of molecules in excited electronic states, e.g. in photofragmentation. In these studies, gas phase UPS or XPS is often combined with mass spectrometry, and particularly detailed information about dynamic processes can be obtained from coincidence measurements, e.g. Photoelectron Photoion Coincidence (PEPICO)[23] .

Experimental results from gas phase XPS are also widely used as reference data for the testing and validation of theoretical methods for modelling core level XPS. For example, core electron binding

energies from gas phase XPS have been used for benchmarking methods for calculating core electron binding energies from first principles, such as the Delta-SCF method and the GW approach[27],[28]. Similarly, core level spectra of free molecules that exhibit strong satellite structures have been used as test cases for more advanced theoretical methods that also permit direct calculation of photoemission satellites. Advantages of using gas phase data for benchmarking purposes include the facts that (i) calculations of free molecules are often computationally more affordable than calculations of extended solids, which is desirable in a benchmarking study when a large number of calculations have to be performed, and (ii) in gas phase XPS, some of the complications that are often encountered in studies of solids, such as contamination of the sample surface, unintentional doping of the sample, or the presence of phase impurities are avoided[28].

2.2 Binding energy databases and data banks

The most extensive and the most widely used database of experimental core electron binding energies is the NIST XPS database[29]. The database was originally published in 1989[30]. Since then, it has been digitized and made freely accessible via a web-based interface, currently hosted at https://srdata.nist.gov/xps/ [31]. The database has been updated several times, but the latest updates were made in September 2012[32]. Currently it contains over 22,000 data entries[9]. Gas phase data are explicitly excluded from the NIST database. For researchers who are interested in gas phase data, a similar resource is currently not available.

The most extensive collection of core electron binding energies from gas phase photoelectron spectroscopy was published by Jolly et al in 1984. This collection contains 2,807 binding energy entries from 189 sources[11]. However, it is only available as a paper copy or a scanned pdf document, and it has not been updated since it was originally published. Nevertheless, even though the data compilation by Jolly et al was published nearly 30 years ago, it has been cited on average ~ 15 times yearly since 2019, indicating that it is still being used as a valuable resource by the community.

2.3 Cheminformatics

The field of research concerned with the collection and organization of chemical data is called cheminformatics[33,34]. In the development of the database of core electron binding energies from gas phase XPS, we will use the tools of cheminformatics to improve the findability of the binding energy data, and to make the database more user-friendly for both direct human users, as well as researchers who employ automated data mining algorithms.

The following concepts from the field of cheminformatics are used in this project:

Molecular formula – The molecular formula indicates the number of atoms of each element present in a molecule. It does not contain any information about the nature of bonding in a molecule or its structure[35]. In this work, the elements in the molecular formulae are listed in alphabetical order; subscripts of 1 are omitted. Examples: C_2H_6O (ethanol), $C_8F_6H_4$ (1,4-bis(trifluoromethyl)benzene).

Chemical formula – The chemical formula is a single typographic line of symbols that contains information about the elemental composition of the molecule, and also some information about the arrangement of the atoms in the molecule[35]. Prior knowledge of common molecular structures and the conventions used when writing chemical formulae is in general required to understand them. Examples: CH_3CH_2OH (ethanol), p-$C_6H_4(CF_3)_2$ (1,4-bis(trifluoromethyl)benzene).

Structural formula – The structural formula is a two-dimensional image that describes the connectivity of the atoms in the molecule, and sometimes also contains additional information about the arrangement of the atoms in three-dimensional space[36,35]. Examples of a structural formula are shown in Figure 3.

(a) (b)

Figure 3: Structural formulae of ethanol and 1,4-bis(trifluoromethyl)benzene.

IUPAC name – The IUPAC name of a molecule is a unique name, assigned to the molecule, that follows the guidelines of chemical nomenclature developed by the International Union of Pure and Applied Chemistry (IUPAC)[37]. In many cases, there is more than one acceptable way in which a molecule can be named[38]. In this work, for each reported binding energy, the version of the name of the molecule that was used in the original publication is generally employed. When there are other commonly used names for the same molecule, those are included as well. Examples: Ethyl Bromoacetate, Trichloro(trifluoromethyl)germane, Aluminum Trifluoroacetylacetonate, and krypton, etc.

InChI – The International Chemical Identifier (InChI) is a string of characters that uniquely identifies a chemical substance[39]. Each InChI string consists of layers of information, including the compulsory main layer describing the molecular formula, the connectivity of the atoms, and the hydrogens present in the molecule, as well as several optional layers including the charge layer, the stereochemical layer, and the isotopic layer[40]. The InChI format was originally developed by the IUPAC and the National Institute of Standards and Technology (NIST), and it is described in [39]. Up-to-date information about the InChI format is available at the website of the InChI trust: www.inchi-trust.org. Examples: InChI=1S/C2H6O/c1-2-3/h3H,2H2,1H3 , InChI=1S/C8H4F6/c9-7(10,11)5-1-2-6(4-3-5)8(12,13)14/h1-4H.

SMILES identifier – The simplified molecular-input line-entry system (SMILES) is an alternative convention for describing a chemical species using a textual identifier[41],[42]. Examples: CCO , C1=CC(=CC=C1C(F)(F)F)C(F)(F)F.

Reference DOI – The digital object identifier (DOI) is a persistent identifier that is used to identify various digital objects, such as journal articles, books and book chapters, data sets, and government reports[43]. DOI-s are widely used in academia in literature references. The new database developed in this project includes the DOI-s of all of the original research articles, where the core electron binding energies included in the database where first reported. Examples: 10.1021/acscentsci.9b00576 , 10.1088/0031-8949/2/1-2/014.

In addition, the following cheminformatics tools were used to collect and organize the data that is stored in the database

RDKit – RDKit is an open source toolkit for cheminformatics. RDKit provides an application programming interface (API) for several different programming languages including Python, Java, C++, and C#. In this project, RDKit has been employed as a Python library[44]. The central concept within RDKit used in this work is the molecule object – this can be manually specified by the user, or automatically generated from a chemical identifier (e.g. the InChI identifier), or from a molecular structure file. In turn, the molecule object can be used to generate the InChI or the SMILES identifier, an image of the structural formula of the molecule, or various other types of data[45]. In this project, RDKit has been used to generate the SMILES identifiers of the molecules, and the molecule images that are included within the database.

jmol – jmol is an open-source viewer for 3D molecular models[46],[47]. An advanced feature of jmol used in this project is the automatic generation of the InChI from the molecular structure. This feature was used to generate the InChI for a molecule, when other means for determining the InChI did not yield a positive result.

OPSIN – OPSIN, an Open Parser for Systematic IUPAC Nomenclature is a web-based tool for parsing IUPAC names of molecules hosted by the University of Cambridge[48],[49]. In this project, OPSIN was used for the automatic generation of the InChI from the name of the molecule, if the InChI could not be straightforwardly retrieved from an online database.

PubChem – PubChem is an online database of molecular data maintained by the National Center for Biotechnology Information, USA. It contains data such as molecular formulae, names, identifiers, structural information, and chemical safety data[50] and other information (where available) for over 60 million unique chemical structures[10]. In this project, PubChem was the first method of choice for determining the InChI of a molecule.

3. Main text

3.1 Structure of the new database

In this project, a new database of core electron binding energies from gas phase photoelectron spectroscopy is constructed. The database consists of a number of database entries – each entry corresponds to one experimentally reported binding energy.

The database entries are numbered sequentially, starting from 1. For each entry, the following types of information are included:

3.1.1 Information about the molecule

Molecular formula – the molecular formula of the molecule that the measurement corresponds to

Chemical formula – the chemical formula is generally presented in the form that was used in the original reference, where the binding energy was reported. For binding energies taken from Jolly et al.,[11], the chemical formula used in [11] is used

Structural formula – an image of the structural formula generated by RDKit is included for each entry

Name – one or several names of the molecule are included. The names to be included are chosen based on perceived likelihood of them being used as a search key by a user of the database – i.e. systematic IUPAC names are not necessarily included in cases where the systematic name is cumbersome and unlikely to be used in practice (e.g. “1,4-Epoxybuta-1,3-diene” for furan).

InChI – the international chemical identifier is included for each molecule

SMILES identifier – likewise, the SMILES identifier is also included in all cases

3.1.2 Information about the reported binding energy

Core level – specifies the element, the atomic shell and subshell, and the total angular momentum quantum number j for non-s levels, e.g. C 1s, Cl $2p_{3/2}$, O 1s, etc.

Binding Energy – the reported binding energy in eV. For binding energies taken from reference Jolly et al. [11], the binding energy value reported in [11] is included. In [11], binding energies reported in the literature were adjusted in cases, where the original publication reported the use of a

reference value for calibrating the binding energy scale, and a different and more accurate reference value for the same reference had been established subsequently.

3.1.3 Bibliographic information

Reference DOI – the DOI of the original reference where the core electron binding energy was reported

3.1.4 Additional information (optional)

Reference text – included for references that do not have a DOI, or where the DOI could not be determined

Reference comment – a comment on the literature reference. For all datapoints taken from Jolly et al. [11], the following comment is included: “Data obtained from Jolly et al., At. Data Nucl. Data Tables 31, 433 (1984). (Reference(s) xxx in Jolly et al.)”

Binding energy comment – a comment on the reported binding energy. E.g. “Str line” / “Wk line” for the strong and weak lines of a multiplet peak.

Comment – a general comment about the data entry. For example, “original reference is wrong, should be Trudell and Price, Canadian Journal of Chemistry 55, 1279 (1977).” / “Error in MF in Jolly et al., Atomic Data and Nuclear Data Tables 31, 433 (1984)”.

3.2 Compiling the initial dataset

At launch, the database is populated with the core electron binding energies (CEBEs) reported in Jolly et al., [11]. A total of 2807 binding energies are reported in [11], and for each of them the following types of information are provided: the molecular formula, the chemical formula, the core level, the binding energy, and a literature reference to the source of the original data.

The job of constructing the new database, and adding the core electron binding energies from Jolly et al to this database can be broken down into the following steps:

(1) Digitizing the data in Jolly et al.

(2) Classifying the data in Jolly et al.

(3) Generating additional data

(4) Organizing the data into an offline database

(5) Developing a web-based user interface for the database

A flow chart depicting the process is shown in Figure 4. Steps 1-3 are discussed in this section. Steps 4-5 are discussed in the next sections.

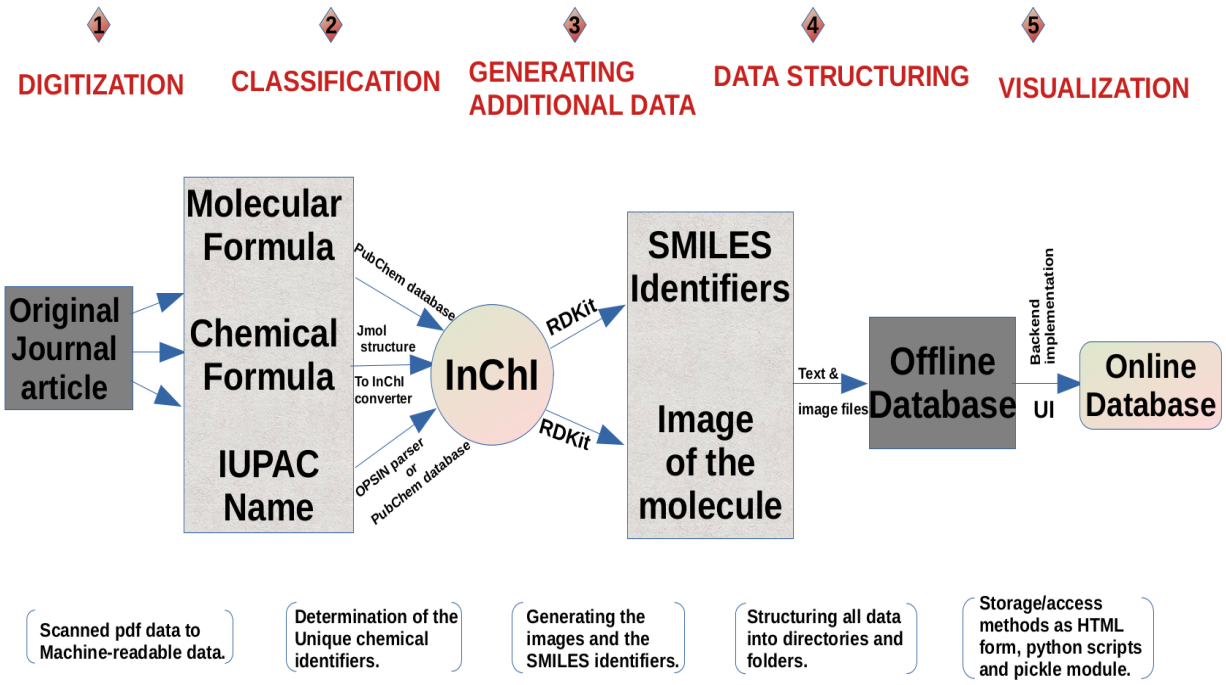


Figure 4. A flow chart describing the conversion of published research data from gas phase photoelectron spectroscopy into an organized, searchable online database of core electron binding energies.

3.2.1 Step 1: digitization

Step (1) involves converting the text data stored in image format in the scanned pdf document of the original article into text data stored as text on a computer. A version of the original article, where optical character recognition (OCR) has been used to recognize the text is available on the publisher's website. However, the OCR text contains many errors, and each string of characters therefore needs to be manually verified. Our preliminary tests with open source OCR software (Tesseract) resulted in similar problems. For example, in the chemical formula $(CH_3)_3GeOC(O)CCl_3$ the OCR software mistakes the lowercase "L" in "Cl" for the number "1", and the formula is instead given as "$(CH_3)_3GeOC(O)CC1_3$". Whilst simple substitution can be used to deal with such problems in conventional written text, in a chemical formula, numbers, letters, and other symbols are all permitted, and therefore editorial oversight by a human reader is still required. It is anticipated that in the near term future, AI-based language models that are also capable of

image processing would be able to perform the task of digitizing the text and correcting for errors of the OCR software reliably, without human intervention.

3.2.2 Step 2: classification

Step (2) is to determine the international chemical identifier (InChI) for the molecule that the reported binding energy corresponds to, and the digital object identifier (DOI) of the source of the original data.

For the DOI lookup, the standard route is to perform a search of the bibliographic reference in Jolly et al in Google Scholar – the correct journal article is usually one of the top results, and the DOI can be retrieved from the publisher's website. In cases where this fails, a back-up option is to manually look up the article in the publisher's catalogue.

For determining the InChI, the preferred route is to perform a search of the molecular formula on the PubChem database, to identify the correct molecule from the list of search results, and to copy the InChI from PubChem. Where this fails, the second option is to use the web-based tool OPSIN to generate the InChI from the IUPAC name of the molecule. Finally, if this approach is also not successful, the last option is to build a 3D model of the molecule in Jmol, and to use Jmol to generate the InChI from the 3D model.

In Jmol, the InChI can be generated using the following command:

```
print {1.1}.find("inchi")
```

To obtain the correct InChI, it is critical that the correct bond orders are specified in the molecular model, otherwise, e.g. if a single bond is specified instead of a double bond, the "missing hydrogens" are automatically added when generating the InChI, and the InChI of the wrong molecule is obtained as a result.

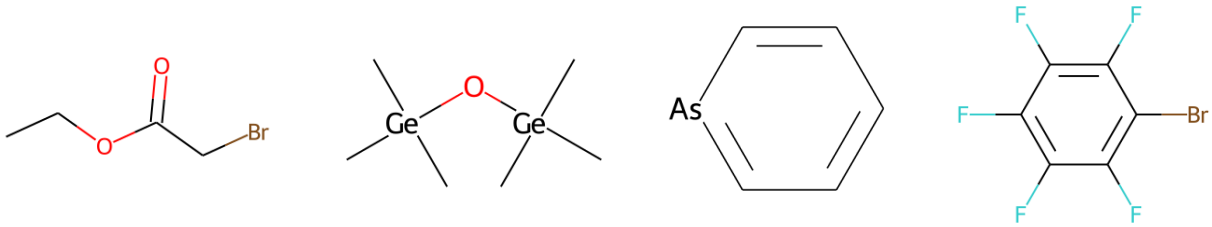
3.2.3 Step 3: generating additional data

In step (3), the InChI of the molecule that the binding energy corresponds to is used to automatically generate additional data for the database. In particular, in this project, RDKit is used to generate the SMILES identifier of the molecule, and an image of the molecule's structural formula.

Version 2023.03.1 of RDKit, installed using Anaconda was used for this project. The commands for generating the structural formula and the SMILES identifier from the InChI are given below.

```
from rdkit import Chem

m = Chem.inchi.MolFromInchi(InChI) or m =
Chem.MolFromSmiles(SMILES)

from rdkit.Chem import AllChem

from rdkit.Chem import Draw

tmp = AllChem.Compute2DCoords(m)

Draw.MolToFile(m, "example.png")

smiles = Chem.MolToSmiles(m)

Chem.MolToSmiles(m)
```

Examples of the images generated using RDKit are shown in Figure 5.



a – $BrCH_2CO_2CH_2CH_3$ b – $((CH_3)_3Ge)_2O$ c – AsC_5H_5 d – BrF_5

Figure 5. Examples of structural formulae in image format generated using RDKit.

3.2.4 The directory structure for the initial dataset

The initial dataset containing the digitized data and the automatically generated data is organized as follows: there is a separate directory for each database entry, containing two files:

- a text file "BE_data" that contains, line by line, the molecular formula, the chemical formula, the name of the molecule, the InChI, the SMILES identifier, the core level, the binding energy, the number of the reference in Jolly et al., the DOI of the reference, and any additional data, such as the reference in text format, a comment about the reported binding energy, or a general comment about the data entry.
- an image file "mol_image", generated by RDKit, with an image of the molecule in .png format. The width and height of each image are 300 and 300 pixels respectively.

3.3 The offline database

In **step (4)**, the initial dataset is converted into a format that is optimized for rapid lookup and efficient storage, and a set of tools is developed to access the information that is stored in the database. Specifically, the offline database consists of the following python lists stored on the disk in binary format:

- MF_list (molecular formulae)
- CF_list (chemical formulae)
- IUPAC_Name_list
- Isomeric_SMILES_+list
- InChI_list
- Core_level_list
- BE_list
- BE_comment_list
- Reference_DOI_list
- Reference_text_list
- Reference_comment_list
- Comment_list

Each list contains the Python object <None> as the first element, and then the value of database entry number n as the n-th element. Thus the lists contain N+1 elements in total, where N is the total number of entries in the database.

The utility pickle is used for writing Python objects to disk as binary files. Storing objects in binary format facilitates rapid input and output. An advantage of using pickle for serialization is that the binary files are compatible across different system architectures and Python releases.

It is noted that in MF_list, the molecular formulae are stored as Python dictionaries with the following structure: {element_1 : num_of_atoms_1 , element_2 : num_of_atoms_2 , …}. A function “parse_molecular_formula” has been written to convert the molecular formula strings into such dictionaries. Storing the molecular formulae in dictionary format is useful for determining, if two molecular formulae are the same, in a way that does not depend on the arbitrary order in which

the elements are specified. i.e. even though the strings “OH2” and “H2O” are not identical, the corresponding dictionary is {“H” : 2 , “O” : 1} in both cases. (In Python dictionaries, the order of the “keys” is arbitrary.)

In addition, the offline database also contains a folder with images corresponding to all entries, named “n.png”, where n is the entry number.

The python program digitized_data_to_database.py, given in Annex 1, is used for converting the digitized data created in step (3) into the offline database. The offline database is publicly available at github.com/UT-XPS/gas-phase-database.

3.4 A web interface for the database

In **step (5)**, a web based user interface for the offline database is built. In the following sections, the design and the implementation of the online database are discussed.

3.4.1 Design of the web interface

The website of the database consists of the following pages:

- The front page (search menu)
- The list of search results
- A detailed view of a single entry

Screenshots of the three pages are shown in Figure 6.

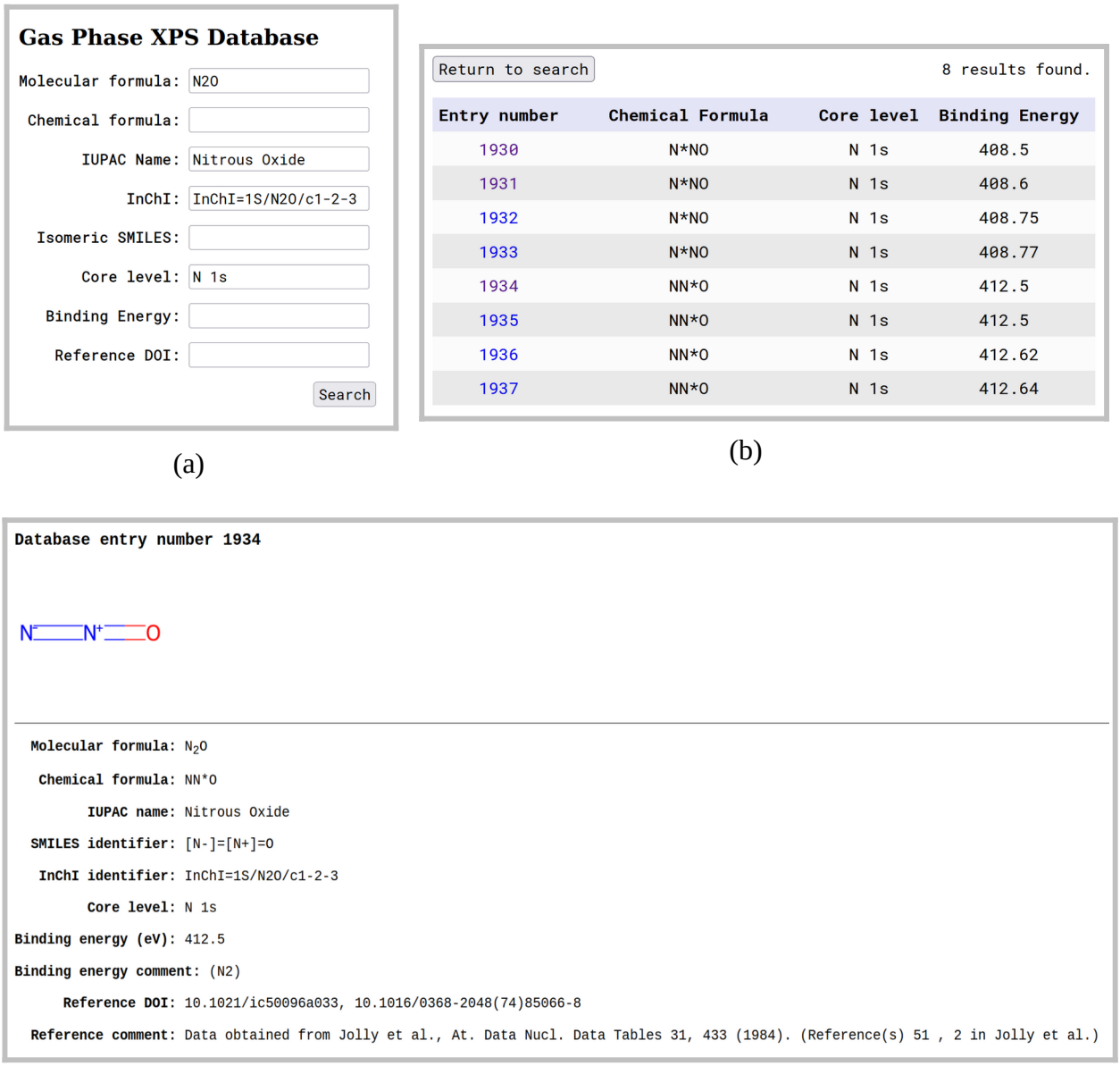


Figure 6: Screenshots of the new gas phase XPS database. (a) The landing page with the search menu, (b) the list of search results, (c) a display of detailed information for one database entry.

The front page of the website, displaying the search menu, contains the following text fields: molecular formula, chemical formula, IUPAC Name, InChI, Isomeric SMILES, Core level, Binding Energy, Reference DOI. The user can enter an input into any number of these text fields, and the information provided by the user is used to search the database. Any text data can be entered into the search fields, except for the “Binding Energy” field that only accepts numerical inputs. If the user tries to search with something other than a number in the Binding Energy field, an error is displayed notifying the user.

Upon pressing the “Search” button, the user is taken to the next page, where the list of search results is displayed. Only the database entries that match all of the search keys provided by the user are displayed. In this page, a summary view of the search results is shown. In particular, for each matching entry, the following bits of information are provided: the entry number, the chemical formula, the core level, and the binding energy. At the top of the page, the number of results found is shown.

In the list of search results, the user can click on any of the entry numbers to access a detailed view of all of the information available for this database entry. In the detailed view page, the database entry number and the image of the structural formula are displayed at the top, followed by the molecular formula, the chemical formula, the name(s) of the molecule, the chemical identifiers, the core level, the binding energy, the DOI of the original article where the binding energy was reported, and any additional comments.

3.4.2 Implementation of the web interface

The top level directory on the server where the database is hosted contains the file index.html, and two directories: “cgi-bin” and “images”.

index.html is a simple html file for the front page of the website with a search menu. The search fields are implemented as an html form. When the form is submitted by clicking the “Search” button, the python script “search_database.py” (see below for details) is called via the common gateway interface (CGI), and the search keys submitted by the user are forwarded to the python script via the “GET” method.

cgi-bin is a directory that contains three python scripts: search_database.py, display_entry.py, and display_index_lines.py. In addition, the actual database, i.e. the pickle files containing the python lists for all categories of information are also stored under cgi-bin.

images is a directory that contains the images of the structural formulae for all database entries.

search_database.py is the script that parses the search terms submitted by the user (~ the query string), and generates the list of search results. It contains a number of function definitions, including functions for parsing the molecular formula submitted by the user (if any), a function for

matching the search terms submitted by the user to database entries, functions for formatting the output, and functions for error handling. In the end, search_database.py generates the html code for the page where the list of search results is displayed.

display_entry.py is the script that generates the html code for the page where a detailed view of one database entry is displayed. It takes the entry number as the only input variable, and based on this, fetches the required information from the binary database files and the image of the molecular formula from the images directory.

display_index_lines.py is a script that generates the html code for the front page of the website with a search menu. display_index_lines.py is invoked, when an error is detected in search_database.py, for example when something that cannot be converted to a float is submitted in the “Binding Energy” field, and it allows the contents of index.html to be displayed, together with an additional error message.

The website is hosted in the University of Tartu webspace, and it can be accessed at xps.ut.ee.

4. Summary

In this project, a new digital database of core electron binding energies from gas phase X-ray Photoelectron Spectroscopy has been built. At launch the database contains over 2,807 binding energy entries. It is anticipated that this database will serve as a useful tool for researchers who work in the field of experimental molecular physics.

All of the data in the database is available for download, either in human-readable format as text data and images, or in a more compact binary format, from github.com/UT-XPS/gas-phase-database. In addition, a web-based user interface for the database has been developed, which is currently hosted at xps.ut.ee. The website has been written in simple html; the dynamic content for the web pages, e.g. the list of search results, is generated using python programs that interface with html via the common gateway interface (CGI).

In the future, we plan to extend the database by continually updating it with new experimental results. In addition, beyond the numerical values of the binding energies, there are also plans to develop the functionality to host actual photoelectron spectra, storing both images of the spectra, as well as the raw x-y data.

5. List of References

(1) *Handbook of X-Ray Photoelectron Spectroscopy: A Reference Book of Standard Spectra for Identification and Interpretation of XPS Data*, Update.; Moulder, J. F., Chastain, J., Eds.; Perkin-Elmer Corporation: Eden Prairie, Minn, 1992.

(2) Hüfner, S. *Photoelectron Spectroscopy: Principles and Applications*, Third revised and enlarged edition.; Springer: New York, 2013.

(3) Watts, J. F. X-Ray Photoelectron Spectroscopy. *Vacuum* **1994**, *45* (6–7), 653–671. https://doi.org/10.1016/0042-207X(94)90107-4.

(4) *Practical Surface Analysis. 1: Auger and x-Ray Photoelectron Spectroscopy*, 2. ed., repr.; Wiley: Chichester, 1996.

(5) Turner, D. W.; Jobory, M. I. A. Determination of Ionization Potentials by Photoelectron Energy Measurement. *The Journal of Chemical Physics* **1962**, *37* (12), 3007–3008. https://doi.org/10.1063/1.1733134.

(6) Engelhard, M. H.; Baer, D. R.; Herrera-Gomez, A.; Sherwood, P. M. A. Introductory Guide to Backgrounds in XPS Spectra and Their Impact on Determining Peak Intensities. *Journal of Vacuum Science & Technology A* **2020**, *38* (6), 063203. https://doi.org/10.1116/6.0000359.

(7) Baer, D. R.; Artyushkova, K.; Richard Brundle, C.; Castle, J. E.; Engelhard, M. H.; Gaskell, K. J.; Grant, J. T.; Haasch, R. T.; Linford, M. R.; Powell, C. J.; Shard, A. G.; Sherwood, P. M. A.; Smentkowski, V. S. Practical Guides for X-Ray Photoelectron Spectroscopy: First Steps in Planning, Conducting, and Reporting XPS Measurements. *Journal of Vacuum Science & Technology A* **2019**, *37* (3), 031401. https://doi.org/10.1116/1.5065501.

(8) Baer, D. R.; Artyushkova, K.; Richard Brundle, C.; Castle, J. E.; Engelhard, M. H.; Gaskell, K. J.; Grant, J. T.; Haasch, R. T.; Linford, M. R.; Powell, C. J.; Shard, A. G.; Sherwood, P. M. A.; Smentkowski, V. S. Practical Guides for X-Ray Photoelectron Spectroscopy: First Steps in Planning, Conducting, and Reporting XPS Measurements. *Journal of Vacuum Science & Technology A* **2019**, *37* (3), 031401. https://doi.org/10.1116/1.5065501.

(9) Powell, C. X-Ray Photoelectron Spectroscopy Database XPS,Version 4.1, NIST Standard Reference Database 20, 1989. https://doi.org/10.18434/T4T88K.

(10) Beamson, G.; Briggs, D. *High Resolution XPS of Organic Polymers: The Scienta ESCA300 Database*; Wiley: Chichester [England] ; New York, 1992.

(11) Jolly, W. L.; Bomben, K. D.; Eyermann, C. J. Core-Electron Binding Energies for Gaseous Atoms and Molecules. *Atomic Data and Nuclear Data Tables* **1984**, *31* (3), 433–493.

https://doi.org/10.1016/0092-640X(84)90011-1.
(12) Hüfner, S. *Photoelectron Spectroscopy: Principles and Applications*, Third revised and enlarged edition.; Springer: New York, 2013.
(13) Hertz, H. Ueber einen Einfluss des ultravioletten Lichtes auf die electrische Entladung. *Ann. Phys. Chem.* **1887**, *267* (8), 983–1000. https://doi.org/10.1002/andp.18872670827.
(14) Jenkin, J. G.; Leckey, R. C. G.; Liesegang, J. The Development of X-Ray Photoelectron Spectroscopy: 1900–1960. *Journal of Electron Spectroscopy and Related Phenomena* **1977**, *12* (1), 1–35. https://doi.org/10.1016/0368-2048(77)85065-2.
(15) Fadley, C. S. X-Ray Photoelectron Spectroscopy: Progress and Perspectives. *Journal of Electron Spectroscopy and Related Phenomena* **2010**, *178–179*, 2–32. https://doi.org/10.1016/j.elspec.2010.01.006.
(16) Stevie, F. A.; Donley, C. L. Introduction to X-Ray Photoelectron Spectroscopy. *Journal of Vacuum Science & Technology A* **2020**, *38* (6), 063204. https://doi.org/10.1116/6.0000412.
(17) Suga, S.; Sekiyama, A.; Tusche, C. A. *Photoelectron Spectroscopy: Bulk and Surface Electronic Structures*, Second edition.; Springer series in surface sciences; Springer: Cham, Switzerland, 2021. https://doi.org/10.1007/978-3-030-64073-6.
(18) Stevie, F. A.; Donley, C. L. Introduction to X-Ray Photoelectron Spectroscopy. *Journal of Vacuum Science & Technology A* **2020**, *38* (6), 063204. https://doi.org/10.1116/6.0000412.
(19) Fadley, C. S. X-Ray Photoelectron Spectroscopy: From Origins to Future Directions. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **2009**, *601* (1–2), 8–31. https://doi.org/10.1016/j.nima.2008.12.189.
(20) Cólon Santana, J. A. *Quantitative Core Level Photoelectron Spectroscopy: A Primer*; IOP concise physics; Morgan & Claypool Publishers: San Rafael, CA, 2015.
(21) Goulielmakis, E.; Loh, Z.-H.; Wirth, A.; Santra, R.; Rohringer, N.; Yakovlev, V. S.; Zherebtsov, S.; Pfeifer, T.; Azzeer, A. M.; Kling, M. F.; Leone, S. R.; Krausz, F. Real-Time Observation of Valence Electron Motion. *Nature* **2010**, *466* (7307), 739–743. https://doi.org/10.1038/nature09212.
(22) Martin, R. M. *Electronic Structure: Basic Theory and Practical Methods*, Second edition.; Cambridge University Press: Cambridge, United Kingdom ; New York, NY, 2020.
(23) Cowan, R. D. *The Theory of Atomic Structure and Spectra*; Los Alamos series in basic and applied sciences; University of California Press: Berkeley, 1981.
(24) *Handbook of X-Ray Photoelectron Spectroscopy: A Reference Book of Standard Spectra for Identification and Interpretation of XPS Data*, Update.; Moulder, J. F., Chastain, J., Eds.; Perkin-

Elmer Corporation: Eden Prairie, Minn, 1992.
(25) Kleimenov, E. High-Pressure X-Ray Photoelectron Spectroscopy Applied to Vanadium Phosphorus Oxide Catalysts under Reaction Conditions. **2005**. https://doi.org/10.14279/DEPOSITONCE-1052.
(26) Brundle, C. R. The Application of Electron Spectroscopy to Surface Studies. *Journal of Vacuum Science and Technology* **1974**, *11* (1), 212–224. https://doi.org/10.1116/1.1318572.
(27) Li, J.; Jin, Y.; Rinke, P.; Yang, W.; Golze, D. Benchmark of *GW* Methods for Core-Level Binding Energies. *J. Chem. Theory Comput.* **2022**, *18* (12), 7570–7585. https://doi.org/10.1021/acs.jctc.2c00617.
(28) Kahk, J. M.; Lischner, J. Accurate Absolute Core-Electron Binding Energies of Molecules, Solids, and Surfaces from First-Principles Calculations. *Phys. Rev. Materials* **2019**, *3* (10), 100801. https://doi.org/10.1103/PhysRevMaterials.3.100801.
(29) Linstrom, P. NIST Chemistry WebBook, NIST Standard Reference Database 69, 1997. https://doi.org/10.18434/T4D303.
(30) Lee, A. Y.; Blakeslee, D. M.; Powell, C. J.; Rumble, Jr., J. R. Development of the Web-Based NIST X-Ray Photoelectron Spectroscopy (XPS) Database. *Data Sci. J.* **2002**, *1*, 1–12. https://doi.org/10.2481/dsj.1.1.
(31) Powell, C. X-Ray Photoelectron Spectroscopy Database XPS,Version 4.1, NIST Standard Reference Database 20, 1989. https://doi.org/10.18434/T4T88K.
(32) Powell, C. X-Ray Photoelectron Spectroscopy Database XPS,Version 4.1, NIST Standard Reference Database 20, 1989. https://doi.org/10.18434/T4T88K.
(33) *Chemoinformatics: A Textbook*; Gasteiger, J., Engels, T., Eds.; Wiley-VCH: Weinheim, 2003.
(34) *Chemoinformatics: A Textbook*; Gasteiger, J., Engels, T., Eds.; Wiley-VCH: Weinheim, 2003.
(35) Flowers, P.; Theopold, K.; Langley, R. *Chemistry*, 2e ed.; OpenStax: Place of publication not identified, 2019.
(36) *Nomenclature of Organic Chemistry: IUPAC Recommendations and Preferred Names 2013*; Favre, H. A., Powell, W. H., International Union of Pure and Applied Chemistry, Eds.; Royal Soc. of Chemistry [u.a.]: Cambridge, 2014.
(37) Favre, H. A.; Powell, W. H. *Nomenclature of Organic Chemistry*; The Royal Society of Chemistry, 2013. https://doi.org/10.1039/9781849733069.
(38) *Principles of Chemical Nomenclature: A Guide to IUPAC Recommendations*, 2011 ed., [2. ed.].; Leigh, G. J., International Union of Pure and Applied Chemistry, Eds.; Royal Society of

Chemistry: Cambridge, 2011.

(39) Heller, S. R.; McNaught, A.; Pletnev, I.; Stein, S.; Tchekhovskoi, D. InChI, the IUPAC International Chemical Identifier. *J Cheminform* **2015**, *7* (1), 23. https://doi.org/10.1186/s13321-015-0068-4.

(40) Heller, S.; McNaught, A.; Stein, S.; Tchekhovskoi, D.; Pletnev, I. InChI - the Worldwide Chemical Structure Identifier Standard. *J Cheminform* **2013**, *5* (1), 7. https://doi.org/10.1186/1758-2946-5-7.

(41) Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28* (1), 31–36. https://doi.org/10.1021/ci00057a005.

(42) Schwaller, P.; Laino, T.; Gaudin, T.; Bolgar, P.; Hunter, C. A.; Bekas, C.; Lee, A. A. Molecular Transformer: A Model for Uncertainty-Calibrated Chemical Reaction Prediction. *ACS Cent. Sci.* **2019**, *5* (9), 1572–1583. https://doi.org/10.1021/acscentsci.9b00576.

(43) *Publication Manual of the American Psychological Association (7th Ed.).*; American Psychological Association: Washington, 2020. https://doi.org/10.1037/0000165-000.

(44) Lowe, D. M.; Corbett, P. T.; Murray-Rust, P.; Glen, R. C. Chemical Name to Structure: OPSIN, an Open Source Solution. *J. Chem. Inf. Model.* **2011**, *51* (3), 739–753. https://doi.org/10.1021/ci100384d.

(45) Molecular Generative Models for Compound Property Prediction. Journal of Chemical Information and Modeling,.

(46) Hanson, R. M. *Jmol* – a Paradigm Shift in Crystallographic Visualization. *J Appl Crystallogr* **2010**, *43* (5), 1250–1260. https://doi.org/10.1107/S0021889810030256.

(47) Hanson, R. M. Jmol SMILES and Jmol SMARTS: Specifications and Applications. *J Cheminform* **2016**, *8* (1), 50. https://doi.org/10.1186/s13321-016-0160-4.

(48) Jessop, D. M.; Adams, S. E.; Murray-Rust, P. Mining Chemical Information from Open Patents. *J Cheminform* **2011**, *3* (1), 40. https://doi.org/10.1186/1758-2946-3-40.

(49) Kim, S.; Chen, J.; Cheng, T.; Gindulyte, A.; He, J.; He, S.; Li, Q.; Shoemaker, B. A.; Thiessen, P. A.; Yu, B.; Zaslavsky, L.; Zhang, J.; Bolton, E. E. PubChem 2019 Update: Improved Access to Chemical Data. *Nucleic Acids Research* **2019**, *47* (D1), D1102–D1109. https://doi.org/10.1093/nar/gky1033.

(50) Kim, S.; Thiessen, P. A.; Bolton, E. E.; Chen, J.; Fu, G.; Gindulyte, A.; Han, L.; He, J.; He, S.; Shoemaker, B. A.; Wang, J.; Yu, B.; Zhang, J.; Bryant, S. H. PubChem Substance and Compound Databases. *Nucleic Acids Res* **2016**, *44* (D1), D1202–D1213. https://doi.org/10.1093/nar/gkv951.

6. Annexes

6.1 Annex 1: contents of digitized_data_to_database.py

```
import os


## This program loads in the digitized data, and converts it into an
organized database.
##
## The original digitized data is organized in folders, one folder per each
binding
## energy entry. In each folder, there are two files: "BE_data" and
"mol_image".
##
## It is assumed that the folder names are numbers from 1 to N, where N is
the total
## number of entries.
##
## BE_data is a text file that has the following general format:
##
## MF: xxx
## CF: xxx
## IUPAC Name: xxx
## Isomeric SMILES: xxx
## InChI: xxx
## Core level: xxx
## BE: xxx
## Reference ID: xxx
## Reference DOI: xxx
## Reference text: xxx
## Comment: xxx
##
## The comment line is optional and may be absent.
##
## mol_image is an image file in .png format, typically created automatically
from
## the InChI using RDKit.
##
## The organized database consists of python lists stored on the disk in
binary
```

```
## format for each category of information:
##
## MF_list (molecular formulae)
## CF_list (chemical formulae)
## IUPAC_Name_list
## Isomeric_SMILES_list
## InChI_list
## Core_level_list
## BE_list
## BE_comment_list
## Reference_DOI_list
## Reference_text_list
## Reference_comment_list
## Comment_list
##
## Each list contains <None> as the first element, and then the value of
database
## entry number n as the n-th element. Thus the lists contain N+1 elements
in
## total, where N is the total number of entries in the database.
##
## Note that there is no Reference_ID_list: the reference ID is the number
of the
## reference ID in Jolly et al., instead, these reference ID-s are stored in
## Reference_comment_list.
##
## BE_comment_list contains any comments that were included with the binding
## energy values in Jolly et al.
##
## In addition, the organized database also contains a folder with images
## corresponding to all entries, named "n.png", where n is the entry number.



## This is the master function that converts the digitized data into the
digital
## database. It does not return anything, but as a result of this function,
the
## database files are written into the specified directory.
##
## Data structures:
```

```
##
## digitized_data_range = list of numbers corresponding to the indices of
the entries,
##                        starting at 1
## all_lists = {"MF" : MF_list, "CF" : CF_list, "IUPAC_Name" :
IUPAC_Name_list, ...}
## MF_list: [None, entry_1_MF_val, entry_2_MF_val, ...]
## Similarly CF_list, IUPAC_Name_list, Isomeric_SMILES_list, and so on
## categories = ["MF", "CF", "IUPAC_Name", "Isomeric_SMILES", "InChI", ...]
## category = string
## database_images_path = string
## digitized_data_path = string - the folder that contains the digitized
data
## database_path = string - the folder that will contain the database files
## entry_number = integer
## digitized_entry_path = string
## BE_data_dict = {"MF" : MF_val, "CF" : CF_val, "IUPAC_Name" :
IUPAC_Name_val, ...}
## All values in BE_data_dict are stored as strings
## original_image_path = string
## new_image_path = string
## list_output_path = string
## f = file
def digitized_data_to_database(digitized_data_path,database_path):
      import os
      import pickle
      digitized_data_range = get_digitized_data_range(digitized_data_path)
      all_lists = {}
      categories = ["MF",
                    "CF",
                    "IUPAC_Name",
                    "Isomeric_SMILES",
                    "InChI",
                    "Core_level",
                    "BE",
                    "BE_comment",
                    "Reference_DOI",
                    "Reference_text",
                    "Reference_comment",
                    "Comment"]
```

```
      for category in categories:
            all_lists[category] = [None]
      database_images_path = database_path + "/images"
      try:
            os.mkdir(database_path)
      except FileExistsError:
            pass
      try:
            os.mkdir(database_images_path)
      except FileExistsError:
            pass
      for entry_number in digitized_data_range:
            print("Working on entry number "+str(entry_number)+".")
            digitized_entry_path = digitized_data_path + "/" +
str(entry_number)
            BE_data_dict = load_BE_data(digitized_entry_path)
            for category in categories:
                  if category == "Reference_comment":


all_lists[category].append(get_reference_comment_from_BE_data_dict(BE_data_d
ict))
                  elif category == "BE":

      all_lists[category].append(get_BE_val_from_BE_data_dict(BE_data_dict))
                  elif category == "BE_comment":


all_lists[category].append(get_BE_comment_from_BE_data_dict(BE_data_dict))
                  elif category == "MF":


all_lists[category].append(parse_molecular_formula(get_value_from_BE_data_di
ct(BE_data_dict,category)))
                  elif category == "IUPAC_Name":


all_lists[category].append(get_value_from_BE_data_dict(BE_data_dict,"IUPAC
Name"))
                  elif category == "Isomeric_SMILES":
```

```
all_lists[category].append(get_value_from_BE_data_dict(BE_data_dict,"Isomeri
c SMILES"))
                elif category == "Core_level":


all_lists[category].append(get_value_from_BE_data_dict(BE_data_dict,"Core
level"))
                elif category == "Reference_DOI":


all_lists[category].append(get_value_from_BE_data_dict(BE_data_dict,"Referen
ce DOI"))
                elif category == "Reference_text":


all_lists[category].append(get_value_from_BE_data_dict(BE_data_dict,"Referen
ce text"))
                else:


all_lists[category].append(get_value_from_BE_data_dict(BE_data_dict,category
))
            original_image_path = digitized_entry_path + "/mol_image"
            new_image_path = database_images_path + "/" + str(entry_number) +
".png"
            os.system("cp " + original_image_path + " " + new_image_path)
      for category in categories:
            list_output_path = database_path + "/" + category + "_list.pkl"
            f = open(list_output_path,"wb")
            pickle.dump(all_lists[category],f)
            f.close()



## Data structures:
##
## digitized_data_range = list of numbers corresponding to the indices of
the entries,
##                        starting at 1
## digitized_data_path = string - the folder that contains the digitized
data
def get_digitized_data_range(digitized_data_path):
      digitized_data_range = []
      dirlist = os.listdir(digitized_data_path)
```

```
      for file_or_folder in dirlist:
            if file_or_folder.isnumeric() == True:
                  digitized_data_range += [file_or_folder]
      digitized_data_range.sort(key=int)
      if int(digitized_data_range[-1]) != len(digitized_data_range):
            print("Warning! There are "+str(len(digitized_data_range))+"
entries, but the index of the last entry is "+str(digitized_data_range[-1])
+".")
      return digitized_data_range



## Data structures:
##
## digitized_entry_path = string
## BE_data_dict = {"MF" : MF_val, "CF" : CF_val, "IUPAC_Name" :
IUPAC_Name_val, ...}
## All values in BE_data_dict are stored as strings
## f = file
## f_lines = [line1, line2, ...]
## each line of f_lines = string
def load_BE_data(digitized_entry_path):
      BE_path = digitized_entry_path + "/BE_data"
      f = open(BE_path,"r")
      f_lines = f.readlines()
      f.close()
      BE_data_dict = {}
      for line in f_lines:
            if line.strip() != "":
                  splitline = line.strip().split(":",1)
                  kkey = splitline[0].strip()
                  if len(splitline) > 1:
                        vvalue = splitline[1].strip()
                  else:
                        vvalue = ""
                  if vvalue != "":
                        BE_data_dict[kkey] = vvalue
                  else:
                        BE_data_dict[kkey] = None
      return BE_data_dict
```

```
## Data structures:
##
## BE_data_dict = {"MF" : MF_val, "CF" : CF_val, "IUPAC_Name" :
IUPAC_Name_val, ...}
## All values in BE_data_dict are stored as strings
## keyword = string
def get_value_from_BE_data_dict(BE_data_dict,keyword):
      if keyword in BE_data_dict.keys():
            return BE_data_dict[keyword]
      else:
            return None


## Data structures:
##
## BE_data_dict = {"MF" : MF_val, "CF" : CF_val, "IUPAC_Name" :
IUPAC_Name_val, ...}
## All values in BE_data_dict are stored as strings
## keyword = string
def get_reference_comment_from_BE_data_dict(BE_data_dict):
      if "Reference ID" in BE_data_dict.keys() and BE_data_dict["Reference
ID"] != None:
            Ref_ID = BE_data_dict["Reference ID"]
            Reference_comment = "Data obtained from Jolly et al., At. Data
Nucl. Data Tables 31, 433 (1984). (Reference(s) " + Ref_ID + " in Jolly et
al.)"
            return Reference_comment
      else:
            return None


## Data structures:
##
## BE_data_dict = {"MF" : MF_val, "CF" : CF_val, "IUPAC_Name" :
IUPAC_Name_val, ...}
## All values in BE_data_dict are stored as strings
## keyword = string
def get_BE_val_from_BE_data_dict(BE_data_dict):
      if "BE" in BE_data_dict.keys():
            if BE_data_dict["BE"] == None:
                  return None
```

```
            BE_splitline = BE_data_dict["BE"].split()
            try:
                  BE_val = float(BE_splitline[0])
            except:
                  BE_val = float(BE_splitline[0].replace(",",".")) 
            return BE_val
      else:
            return None

## Data structures:
##
## BE_data_dict = {"MF" : MF_val, "CF" : CF_val, "IUPAC_Name" :
IUPAC_Name_val, ...}
## All values in BE_data_dict are stored as strings
## keyword = string
def get_BE_comment_from_BE_data_dict(BE_data_dict):
      if "BE" in BE_data_dict.keys():
            if BE_data_dict["BE"] == None:
                  return None
            BE_splitline = BE_data_dict["BE"].split()
            if len(BE_splitline) > 1:
                  BE_comment = ""
                  for substring in BE_splitline[1:]:
                        BE_comment += (substring+" ")
                  return BE_comment.strip()
            else:
                  return None
      else:
            return None


## This function parses a molecular formula string, and returns a dictionary
with
## the elements present in the molecule as the keys, and the number of atoms
of each
## element as values. A new element is assumed to start every time there is
a
## capital letter.
##
## Data structures:
```

```
##
## MF_val = string
## MF_list = ["Symbol1numatoms1", "Symbol2numatoms2", ...], e.g. ["C2",
"H6"]
## MF_dict = {element_1 : num_of_atoms_1 , element_2 : num_of_atoms_2 , ...}
def parse_molecular_formula(MF_val):
      MF_list = []
      symbol_and_num = ""
      if MF_val == "" or MF_val == None:
            return None
      else:
            for character in MF_val:
                  if character.isupper() == True:
                        if symbol_and_num != "":
                              MF_list += [symbol_and_num]
                        symbol_and_num = character
                  else:
                        symbol_and_num += character
      if symbol_and_num != "":
            MF_list += [symbol_and_num]
      MF_dict = {}
      for symbol_and_num in MF_list:
            current_symbol = ""
            current_quantity = ""
            for character in symbol_and_num:
                  if character.isalpha() == True:
                        current_symbol += character
                  elif character.isnumeric() == True:
                        current_quantity += character
            if current_quantity != "":
                  current_quantity = int(current_quantity)
            else:
                  current_quantity = 1
            MF_dict[current_symbol] = current_quantity
      return MF_dict
```

6.2 Annex 2: contents of index.html

```
<head>
```

```
<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link href="https://fonts.googleapis.com/css2?
family=Roboto+Mono&display=swap" rel="stylesheet">

<meta name="viewport" content="width=device-width, initial-scale=0.8">

</head>

<h2> Gas Phase XPS Database </h2>

<form action = "/cgi-bin/search_database.py" method = "get">

<p style='font-family: "Roboto Mono", monospace'>  <b> Molecular formula:
<input type = "text" name = "MF" style='font-family: "Roboto Mono",
monospace; font-size : 16px'> </p>

<p style='font-family: "Roboto Mono", monospace'>  &nbspChemical formula:
<input type = "text" name = "CF" style='font-family: "Roboto Mono",
monospace; font-size : 16px'> </p>

<p style='font-family: "Roboto Mono", monospace'>
      &nbspIUPAC Name:  <input type = "text" name =
"IUPAC_Name" style='font-family: "Roboto Mono", monospace; font-size :
16px'> </p>

<p style='font-family: "Roboto Mono", monospace'>
           &nbspInChI:  <input
type = "text" name = "InChI" style='font-family: "Roboto Mono", monospace;
font-size : 16px'>  </p>

<p style='font-family: "Roboto Mono", monospace'>   &nbspIsomeric
SMILES:  <input type = "text" name = "Isomeric_SMILES" style='font-family:
"Roboto Mono", monospace; font-size : 16px'> </p>

<p style='font-family: "Roboto Mono", monospace'>
      &nbspCore level:  <input type = "text" name =
"Core_level" style='font-family: "Roboto Mono", monospace; font-size :
16px'> </p>

<p style='font-family: "Roboto Mono", monospace'>    &nbspBinding
Energy:  <input type = "text" name = "BE" style='font-family: "Roboto Mono",
monospace; font-size : 16px'>  </p>

<p style='font-family: "Roboto Mono", monospace'>
   &nbspReference DOI: </b> <input type = "text" name =
"Reference_DOI" style='font-family: "Roboto Mono", monospace; font-size :
16px'> </p>

<p style='font-family: "Roboto Mono", monospace'>
        

        

        

        <input type = "submit" style='font-family:
"Roboto Mono", monospace; font-size : 16px' value = "Search" /> </p>

</form>
```

6.3 Annex 3: contents of search_database.py

```
#!/usr/bin/python3
# Read in the environment variable created by HTML

import sys
import os
import pickle
from urllib.parse import unquote
from urllib.parse import quote_plus

os.chdir("..")

GET = {}
QUERY_STRING = os.getenv("QUERY_STRING")
args = QUERY_STRING.split('&')

for arg in args:
      t = arg.split('=')
      if len(t) > 1:
            kkey = arg.split('=')[0]
            vvalue = arg.split('=')[1]
            vvalue = vvalue.replace('+',' ')
            vvalue = unquote(vvalue)
            GET[kkey] = vvalue

# Get data from fields
MF = GET['MF']
CF = GET['CF']
IUPAC_Name = GET['IUPAC_Name']
InChI = GET['InChI']
Isomeric_SMILES = GET['Isomeric_SMILES']
Core_level = GET['Core_level']
BE = GET['BE']
Reference_DOI = GET['Reference_DOI']

# Perform tasks using data

def load_pkl(category_filename):
```

```
	f = open(category_filename,"rb")
	category_list = pickle.load(f)
	f.close()
	return category_list


## This function parses a molecular formula string, and returns a dictionary with
## the elements present in the molecule as the keys, and the number of atoms of each
## element as values. A new element is assumed to start every time there is a
## capital letter.
##
## Data structures:
##
## MF_val = string
## MF_list = ["Symbol1numatoms1", "Symbol2numatoms2", ...], e.g. ["C2", "H6"]
## MF_dict = {element_1 : num_of_atoms_1 , element_2 : num_of_atoms_2 , ...}
def parse_molecular_formula(MF_val):
	MF_list = []
	symbol_and_num = ""
	if MF_val == "" or MF_val == None:
		return None
	else:
		for character in MF_val:
			if character.isupper() == True:
				if symbol_and_num != "":
					MF_list += [symbol_and_num]
				symbol_and_num = character
			else:
				symbol_and_num += character
	if symbol_and_num != "":
		MF_list += [symbol_and_num]
	MF_dict = {}
	for symbol_and_num in MF_list:
		current_symbol = ""
		current_quantity = ""
		for character in symbol_and_num:
			if character.isalpha() == True:
```

```
                                current_symbol += character
                        elif character.isnumeric() == True:
                                current_quantity += character
                if current_quantity != "":
                        current_quantity = int(current_quantity)
                else:
                        current_quantity = 1
                MF_dict[current_symbol] = current_quantity
        return MF_dict


def add_subscripts_to_CF_string(CF_val):
      string_with_subscripts = ""
      pre_previous_character = None
      previous_character = None
      for current_character in CF_val:
            if current_character.isdigit() and previous_character not in ["
","-","_",",","(",";",None]:
                  if previous_character.isdigit() == False:
                        string_with_subscripts +=
"<sub>"+current_character+"</sub>"
                  elif previous_character.isdigit() and
pre_previous_character not in [" ","-","_",",","(",";",None]:
                        string_with_subscripts +=
"<sub>"+current_character+"</sub>"
                  else:
                        string_with_subscripts += current_character
            else:
                  string_with_subscripts += current_character
            pre_previous_character = previous_character
            previous_character = current_character
      return string_with_subscripts


def get_entries_that_match_all_search_terms(GET):
      categories =
['MF','CF','IUPAC_Name','InChI','Isomeric_SMILES','Core_level','BE','Referen
ce_DOI']
      sets_of_indices = []
      for category in categories:
```

```
            search_term = GET[category]
            if search_term.strip() != "":
                  if category == "MF":
                        search_term = parse_molecular_formula(search_term)
                  if category == "BE":
                        try:
                              search_term = float(search_term)
                        except:
                              error_message = "Cannot convert \'" +
search_term + "\' to a float. The binding energy must be a number."
                              back_to_index(error_message)
                              sys.exit(0)
                  category_filename = "cgi-bin/"+category+"_list.pkl"
                  category_list = load_pkl(category_filename)
                  matching_indices = set()
                  i = 0
                  for entry in category_list:
                        if category == "BE":
                              if type(entry) == float:
                                    try:
                                          if abs(search_term - entry) < 0.001:
                                                matching_indices.add(i)
                                    except:
                                          pass
                        elif category == "CF":
                              if type(entry) == str:
                                    if (search_term == entry) or (search_term
== entry.replace("*","")):
                                          matching_indices.add(i)
                        elif category == "IUPAC_Name":
                              if type(entry) == str:
                                    split_name = entry.split(",")
                                    for name in split_name:
                                          if search_term.casefold() ==
name.strip().casefold():
                                                matching_indices.add(i)
                        elif search_term == entry:
                              matching_indices.add(i)
                        i += 1
                  sets_of_indices += [matching_indices]
```

```
      entries_that_match_all_search_terms =
set.intersection(*sets_of_indices)
      return entries_that_match_all_search_terms



def back_to_index(error_message):
      error_line = "<p style='font-family: Courier New ; color: red'>" +
error_message + "</p>"
      index_file = open("index.html","r")
      index_lines = index_file.readlines()
      index_file.close()
      print("Content-type:text/html\r\n\r\n")
      print("<html>")
      print("<head>")
      print("<title>Hello - Second CGI Program</title>")
      print("</head>")
      print("<body>")
      for line in index_lines:
            if 'name = ' in line:
                  splitline = line.split("name = ")
                  kkey = splitline[1].split()[0][1:-1]
                  vvalue = GET[kkey]
                  newline = splitline[0]+'value = "'+vvalue+'" name =
'+splitline[1]
                  print(newline)
            else:
                  print(line)
      print(error_line)
      print("</body>")
      print("</html>")
      sys.exit(1)

def get_data_for_summary_table(set_of_entries):
      data_for_summary_table = {}
      for entry_number in set_of_entries:
            data_for_summary_table[entry_number] = {}
      for category in ['CF','Core_level','BE']:
            category_filename = "cgi-bin/"+category+"_list.pkl"
            category_list = load_pkl(category_filename)
            for entry_number in set_of_entries:
```

```
                value = category_list[entry_number]
                data_for_summary_table[entry_number][category] = value
    return data_for_summary_table


def prepare_line(data_for_summary_table,entry_number,i):
    raw_entry_number = str(entry_number)
    raw_Core_level = data_for_summary_table[entry_number]["Core_level"]
    raw_BE = str(data_for_summary_table[entry_number]["BE"])
    raw_CF = data_for_summary_table[entry_number]["CF"]
    BE_string = raw_BE.center(16).replace(" "," ")
    if len(raw_CF) <= 24:
        entry_number_string = raw_entry_number.center(12).replace("
"," ")
        Core_level_string = raw_Core_level.center(12).replace("
"," ")
        CF_string_length = 24
        CF_fontsize = 16
        CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
    elif len(raw_CF) <= 28:
        entry_number_string = raw_entry_number.center(12)[:-3].replace("
"," ")
        Core_level_string = raw_Core_level.center(12)[1:].replace("
"," ")
        CF_string_length = 28
        CF_fontsize = 16
        CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
    elif len(raw_entry_number) <= 3 and len(raw_CF) <= 29:
        entry_number_string = raw_entry_number.center(12)[:-4].replace("
"," ")
        Core_level_string = raw_Core_level.center(12)[1:].replace("
"," ")
        CF_string_length = 29
        CF_fontsize = 16
        CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
    elif len(raw_Core_level) <= 5 and len(raw_CF) <= 29:
        entry_number_string = raw_entry_number.center(12)[:-3].replace("
"," ")
```

```
            Core_level_string = raw_Core_level.center(12)[2:].replace("
"," ")
            CF_string_length = 29
            CF_fontsize = 16
            CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
      elif len(raw_entry_number) <= 3 and len(raw_Core_level) <= 5 and
len(raw_CF) <= 30:
            entry_number_string = raw_entry_number.center(12)[:-4].replace("
"," ")
            Core_level_string = raw_Core_level.center(12)[2:].replace("
"," ")
            CF_string_length = 30
            CF_fontsize = 16
            CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
      elif len(raw_Core_level) <= 4 and len(raw_CF) <= 30:
            entry_number_string = raw_entry_number.center(12)[:-3].replace("
"," ")
            Core_level_string = raw_Core_level.center(12)[3:].replace("
"," ")
            CF_string_length = 30
            CF_fontsize = 16
            CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
      elif len(raw_entry_number) <= 3 and len(raw_Core_level) <= 4 and
len(raw_CF) <= 31:
            entry_number_string = raw_entry_number.center(12)[:-4].replace("
"," ")
            Core_level_string = raw_Core_level.center(12)[3:].replace("
"," ")
            CF_string_length = 31
            CF_fontsize = 16
            CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
      elif len(raw_CF) <= 36:
            entry_number_string = raw_entry_number.center(12)[:-2].replace("
"," ")
            Core_level_string = raw_Core_level.center(12)[1:].replace("
"," ")
            CF_string_length = 36
            CF_fontsize = 12
```

```
            CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
      elif len(raw_CF) <= 40:
            entry_number_string = raw_entry_number.center(12)[:-1].replace("
"," ")
            Core_level_string = raw_Core_level.center(12).replace("
"," ")
            CF_string_length = 40
            CF_fontsize = 10
            CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
      else:
            entry_number_string = raw_entry_number.center(12)[:-1].replace("
"," ")
            Core_level_string = raw_Core_level.center(12).replace("
"," ")
            CF_string_length = 40
            CF_fontsize = 10
            raw_CF = raw_CF[:37]+"..."
            CF_string =
prepare_CF_string(raw_CF,CF_string_length,CF_fontsize)
      if (-1)**i == -1:
            line_start = '''<p style='font-family: \"Roboto Mono\", Courier,
monospace; font-size: 16 px ; padding: 0px ; margin : 0px'> <span
style='background-color : rgb(233,233,233) ; font-size: 16px ; padding:
6px'> <nobr> <a href='display_entry.py?entry_number='''
      else:
            line_start = '''<p style='font-family: \"Roboto Mono\", Courier,
monospace; font-size: 16 px ; padding: 0px ; margin : 0px'> <span
style='background-color : rgb(251,251,251) ; font-size: 16px ; padding:
6px'> <nobr> <a href='display_entry.py?entry_number='''
      line = line_start + str(entry_number) + "' style='text-
decoration:none'>" + entry_number_string + "</a> " + CF_string +
Core_level_string + BE_string + " </span> </nobr> </p>"
      return line


def prepare_CF_string(raw_CF,CF_string_length,CF_fontsize):
      CF_string =
add_subscripts_to_CF_string(raw_CF.center(CF_string_length).replace("
"," "))
      subscript_count = CF_string.count("<sub>")
      CF_string_padding = "<sup>"
      for j in range(subscript_count):
            CF_string_padding += " "
```

```
      CF_string_padding += "</sup>"
      CF_string = CF_string_padding+CF_string+CF_string_padding
      CF_string = "<span style=\'font-size: " + str(CF_fontsize) + "px\'>" +
CF_string + "</span>"
      return CF_string

##########################################################################
####################

entries_that_match_all_search_terms =
get_entries_that_match_all_search_terms(GET)
data_for_summary_table =
get_data_for_summary_table(entries_that_match_all_search_terms)
sorted_list_of_entry_numbers =
sorted(list(entries_that_match_all_search_terms))

# Return data to print

print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head>")
print("<title>Hello - Second CGI Program</title>")
print('''<link rel="preconnect" href="https://fonts.googleapis.com">''')
print('''<link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>''')
print('''<link href="https://fonts.googleapis.com/css2?
family=Roboto+Mono:wght@400;700&display=swap" rel="stylesheet">''')
print("<style>")
print("span {display : inline-block}")
print("sub {font-size : 75%}")
print("sup {font-size : 12.5%}")
print("p {-webkit-text-size-adjust: 100%;}")
print("</style>")
print("<meta name='viewport' content='width=660, initial-scale=0.55'>")
print("</head>")
print("<body>")
#print("<p>" + os.getcwd() + "</p>")
num_of_results = len(sorted_list_of_entry_numbers)
first_line = "<p style='font-family: \"Roboto Mono\", Courier, monospace;
font-size : 15px'> <nobr> <a href=\'display_index_lines.py?error_message=&"
+ QUERY_STRING + "\'> <button style='font-family: \"Roboto Mono\", Courier,
monospace; font-size : 15px'> Return to search</button></a> "
```

```
for i in range(37-len(str(num_of_results))):
      first_line += " "
if num_of_results == 1:
      first_line += str(num_of_results) + " result found. </nobr>
</p>"
else:
      first_line += str(num_of_results) + " results found.</nobr>
</p>"
print(first_line)
print("<p style='font-family: \"Roboto Mono\", Courier, monospace; font-
size: 16px ; padding: 0px ; margin : 0px'> <span style='background-color :
rgb(228,228,247) ; font-size: 16px ; padding: 6px'> <b>
Entry number    &nbspChemical Formula  &n
bsp  Core level  Binding Energy  </b>
</span> </p>")
i = 0
for entry_number in sorted_list_of_entry_numbers:
      line = prepare_line(data_for_summary_table,entry_number,i)
      print(line)
      i += 1
print("</body>")
print("</html>")


sys.exit(0)
```

6.4 Annex 4: contents of display_entry.py

```
#!/usr/bin/python3


import os
import pickle
from urllib.parse import unquote


os.chdir("..")


GET = {}
args = os.getenv("QUERY_STRING").split('&')


for arg in args:
        t = arg.split('=')
```

```
        if len(t) > 1:
                kkey = arg.split('=')[0]
                vvalue = arg.split('=')[1]
                vvalue = vvalue.replace('+',' ')
                vvalue = unquote(vvalue)
                GET[kkey] = vvalue

def load_pkl(category_filename):
        f = open(category_filename,"rb")
        category_list = pickle.load(f)
        f.close()
        return category_list

def add_subscripts_to_CF_string(CF_val):
      string_with_subscripts = ""
      pre_previous_character = None
      previous_character = None
      for current_character in CF_val:
            if current_character.isdigit() and previous_character not in ["
","-","_",",","(",";",None]:
                  if previous_character.isdigit() == False:
                        string_with_subscripts +=
"<sub>"+current_character+"</sub>"
                  elif previous_character.isdigit() and
pre_previous_character not in [" ","-","_",",","(",";",None]:
                        string_with_subscripts +=
"<sub>"+current_character+"</sub>"
                  else:
                        string_with_subscripts += current_character
            else:
                  string_with_subscripts += current_character
            pre_previous_character = previous_character
            previous_character = current_character
      return string_with_subscripts

def make_MF_human_readable(MF_dict):
      MF_string = ""
      keyys = sorted(list(MF_dict.keys()))
      for keyy in keyys:
            MF_string += keyy
```

```
            if MF_dict[keyy] != 1:
                  MF_string += "<sub>"
                  MF_string += str(MF_dict[keyy])
                  MF_string += "</sub>"
      return MF_string

def get_img_size(MF_dict):
      num_atoms = 0
      for element in MF_dict.keys():
            number = MF_dict[element]
            num_atoms += number
      if num_atoms <= 5:
            img_size = {"mobile": 40, "desktop": 180}
      elif num_atoms <= 12:
            img_size = {"mobile": 55, "desktop": 240}
      else:
            img_size = {"mobile": 70, "desktop": 300}
      return img_size

categories = ["MF",
            "CF",
            "IUPAC_Name",
            "Isomeric_SMILES",
            "InChI",
            "Core_level",
            "BE",
            "BE_comment",
            "Reference_DOI",
            "Reference_text",
            "Reference_comment",
            "Comment"]

category_display_names = {"MF" : "Molecular formula",
                  "CF" : "Chemical formula",
                  "IUPAC_Name" : "IUPAC name",
                  "Isomeric_SMILES" : "SMILES identifier",
                  "InChI" : "InChI identifier",
                  "Core_level" : "Core level",
                  "BE" : "Binding energy (eV)",
```

```
                "BE_comment" : "Binding energy comment",
                "Reference_DOI" : "Reference DOI",
                "Reference_text" : "Reference (text)",
                "Reference_comment" : "Reference comment",
                "Comment" : "Comment"}

entry_number = int(GET['entry_number'])
all_values = {}
for category in categories:
      category_filename = "cgi-bin/"+category+"_list.pkl"
      category_list = load_pkl(category_filename)
      category_value = category_list[entry_number]
      if category == "CF":
            category_value = add_subscripts_to_CF_string(category_value)
      elif category == "MF":
            img_size = get_img_size(category_value)
            category_value = make_MF_human_readable(category_value)
      all_values[category] = category_value

print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head>")
print("<title>Hello - Second CGI Program</title>")
print("<style>")
print("@media (max-width:1000px) { img#desktopver {display: none;} }")
print("@media (min-width:1001px) { img#mobilever {display: none;} }")
print("</style>")
print("</head>")
print("<body>")
print("<h3 style='font-family: Courier New ; padding: 0px ; margin : 0px'>
Database entry number "+str(entry_number)+"</h3>")
print("<img src='../images/"+str(entry_number)+".png' style='max-width: " +
str(img_size['desktop']) + "px' id='desktopver'>")
print("<img src='../images/"+str(entry_number)+".png' style='width: " +
str(img_size['mobile']) + "vw' id='mobilever'>")
print("<hr>")
for category in categories:
      if all_values[category] != None:
```

```
            print("<p style='font-family: Courier New'>
<b>"+category_display_names[category].rjust(19).replace(" "," ")+":</b>
"+str(all_values[category])+" </p>")

print("</body>")

print("</html>")
```

7. License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Daniel Agbonluai Ijegbai,

(*author's name*)

1. grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

"An Open-Access Online Database of Core Electron Binding Energies from Gas Phase Photoelectron Spectroscopy",

(*title of thesis*)

supervised by Dr Juhan Matthias Kahk.

(*supervisor's name*)

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in points 1 and 2.
4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Daniel Agbonluai Ijegbai

29/05/2023