UNIVERSITÀ DI PAVIA

FACOLTÀ DI INGEGNERIA
DIPARTIMENTO DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN BIOINGEGNERIA

TESI DI LAUREA

A FRAMEWORK FOR GENETIC VARIANT VALIDATION: DESIGN, IMPLEMENTATION AND ASSESSMENT

PROGETTAZIONE, IMPLEMENTAZIONE E VALUTAZIONE DI UN FRAMEWORK PER LA VALIDAZIONE DELLE VARIANTI GENOMICHE

Candidato: Edoardo Giani

Relatore: Chiar.mo Prof. Paolo Magni

Correlatore: Silvia Berardelli, PhD
Ing. Susanna Zucca, PhD

Anno Accademico 2024/2025

But how could you live
and have no story to tell?
Fëdor Dostoevskij,
White Nights

Abstract

The increasing adoption of Next-Generation Sequencing (NGS) technologies has led to a massive increase in the number of detected genetic variants, making their correct interpretation a central challenge in modern genomics. In this context, validation of variant nomenclature plays a key role in ensuring consistent and unambiguous representation of genetic variants across different reference systems and nomenclatures.

However, existing validation tools present several limitations, such as reliance on external services, incomplete support for different variant representations, and ambiguity in the handling of heterogeneous inputs. These issues can negatively impact uniformity and complicate integration within automated analysis pipelines.

The work presented in this thesis addresses these challenges by investigating existing variant validation tools and developing an improved validation framework designed to operate locally, handle multiple input formats, and reduce ambiguity in variant representation. The work includes a comparative analysis of state-of-the-art tools, the extension of a selected variant nomenclature validation framework to overcome identified limitations, and a systematic evaluation of the proposed improvements using two different benchmark datasets.

The resulting approach could also be integrated into a real-world application for genomic variant interpretation, where it could be used to validate and normalize user inputs. The proposed framework improves the completeness and consistency of variant representations while maintaining agreement with well-established state-of-the-art tools. This work, carried out in collaboration with enGenome srl (a spin-off of the University of Pavia), contributes to the development of more reliable and robust variant validation strategies, supporting their use in both research and applied genomic contexts.

Prefazione

La crescente diffusione delle tecnologie di Next-Generation Sequencing (NGS) ha portato a un aumento significativo del numero di varianti genetiche identificate, rendendo la loro corretta interpretazione una sfida centrale nella genomica moderna. In questo contesto, la validazione delle varianti svolge un ruolo fondamentale nel garantire una rappresentazione coerente e univoca delle varianti tra diversi sistemi di riferimento e nomenclature.

Tuttavia, gli strumenti di validazione attualmente disponibili presentano diverse limitazioni, tra cui la dipendenza da servizi esterni, supporto incompleto per le diverse rappresentazioni delle varianti e ambiguità nella gestione di input eterogenei. Questi aspetti possono compromettere l'uniformità dei risultati e complicare l'integrazione all'interno di pipeline di analisi automatizzate.

Il lavoro presentato in questa tesi affronta tali problematiche attraverso l'analisi degli strumenti di validazione esistenti e lo sviluppo di un framework migliorato, progettato per operare localmente, gestire molteplici formati di input e ridurre le ambiguità nella rappresentazione delle varianti. Il lavoro comprende un'analisi comparativa degli strumenti allo stato dell'arte, l'estensione di un framework di validazione selezionato per superarne le limitazioni individuate e una valutazione sistematica dei miglioramenti proposti utilizzando due distinti dataset di benchmark.

L'approccio risultante può inoltre essere integrato in un'applicazione reale per l'interpretazione di varianti genomiche, dove potrebbe essere utilizzato per validare e normalizzare gli input forniti dagli utenti. Il framework proposto migliora la completezza e la coerenza delle rappresentazioni delle varianti, mantenendo al contempo un'elevata concordanza con gli strumenti di riferimento più consolidati.

Questo lavoro, svolto in collaborazione con enGenome srl (spin-off dell'Università di Pavia), contribuisce allo sviluppo di strategie di validazione delle varianti più affidabili e robuste, favorendone l'impiego sia in ambito di ricerca che in contesti applicativi della genomica.

Contents

List of Figures

List of Tables

Acronyms

DNA	Deoxyribonucleic Acid
gDNA	Genomic DNA
cDNA	Complementary DNA
RNA	Ribonucleic Acid
mRNA	Messenger RNA
NGS	Next-Generation Sequencing
SNV	Single Nucleotide Variant
SNP	Single Nucleotide Polymorphism
InDel	Insertion/Deletion
MNV	Multi-Nucleotide Variant
SV	Structural Variant
CNV	Copy Number Variant
CDS	Coding Sequence
UTR	Untranslated Region
HGVS	Human Genome Variation Society
Ensembl	Ensembl Genome Database

RefSeq	Reference Sequence Database
LRG	Locus Reference Genomic
UCSC	University of California, Santa Cruz Genome Browser
GRC	Genome Reference Consortium
VCF	Variant Call Format
dbSNP	Database of Single Nucleotide Polymorphisms
rsID	Reference SNP cluster ID
NCBI	National Center for Biotechnology Information
GTF	Gene Transfer Format
GFF	General Feature Format
VEP	Variant Effect Predictor
MANE	Matched Annotation from NCBI and Ensembl
API	Application Programming Interface
REST	Representational State Transfer
JSON	JavaScript Object Notation

Chapter 1

Introduction

1.1 DNA mutations

A genetic mutation is an alteration in the DNA sequence relative to a specific region of a genome. It arises from replication errors, principally during cellular division, when genetic material is replicated and transmitted to daughter cells, and may affect genes or chromosomes. Mutations may be caused also by spontaneous chemical changes or induced by chemical or physical genotoxic agents. Since DNA encodes molecular instructions that govern cellular structure and function, such alterations can influence biological processes at multiple levels.

Although genetic mutations can contribute to the development of pathological conditions, such as cancer, a significant number is thought to be unharmful, that is, it does not affect the individual. A genetic mutation may even be beneficial and consequently tend to spread rapidly through a specific population, while deleterious changes tend to die along with the organism that hosts them [1].

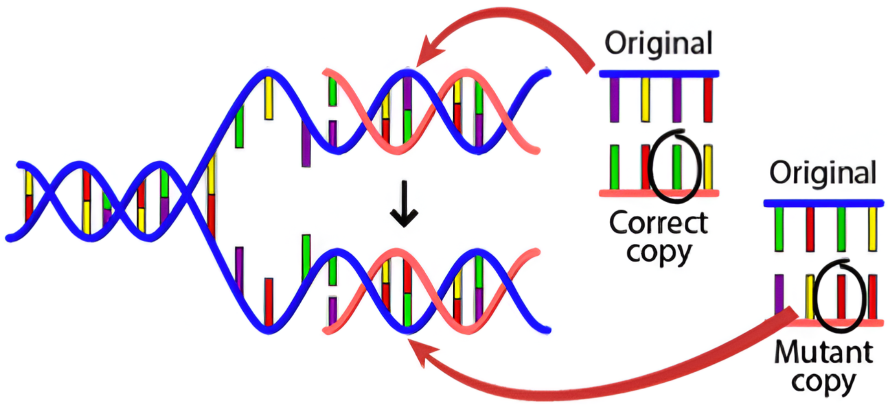

Figure 1.1: **DNA replication error leading to mutation.** The process of DNA replication may lead to an incorrect nucleotide incorporation event. While one daughter strand represents the correct copy of the original sequence, the other contains a mismatched base, generating a mutant copy. If not repaired by cellular mechanisms, such replication errors can become permanent mutations in subsequent cell divisions. Source: [1].

In the context of human genetics, the terms *mutation* and *variant* are often used interchangeably to indicate a DNA sequence change relative to a reference genome. However, the term *variant* is generally preferred in modern genomics, as it does not imply any assumption about pathogenicity.

At the population level, every human genome carries a significant number of DNA changes, and their distribution and effects vary widely across individuals [2]. This population-dependent variability is illustrated in Figure 1.2, which summarizes the distribution and sharing of genetic variants across global populations.

While the vast majority of these variants are neutral or have minimal functional impact, a small fraction may alter gene function in a biologically significant manner. When such deleterious variants affect a single gene and are sufficient to cause a pathological phenotype, they give rise to monogenic, or Mendelian, diseases [3].

Even though each condition is typically rare when considered separately, monogenic disorders collectively affect a large number of individuals worldwide with an estimated prevalence of 7% to 8% in the United States [4]. Furthermore, monogenic diseases are often associated with severe clinical phenotypes that have high morbidity and mortality rates [4].

Examples of monogenic disorders with their approximate occurrence rates are shown in Table 1.1, highlighting the strong variability between diseases and populations.

Table 1.1: Selected examples of monogenic diseases and approximate occurrence rates. Adapted from [5].

Condition	**Approximate occurrence rate**
Achondroplasia	1 in 15,000 to 40,000 live births worldwide
Beta-thalassemia	Most prevalent in populations from the Mediterranean, North Africa, the Middle East, South and Central Asia, and Southeast Asia (˜30 per 100,000 newborns)
Cystic fibrosis	1 in 2,500–3,500 White Americans; 1 in 17,000 African Americans; 1 in 31,000 Asian Americans
Fragile X syndrome	1 in 4,000 males; 1 in 8,000 females
Huntington disease	3–7 per 100,000 individuals of European descent; less common in Japanese, Chinese, and African descent
Sickle cell disease	>100,000 individuals in the US; ˜1 in 365 newborn African Americans; ˜1 in 16,300 Hispanic Americans
Hemophilia	Predominantly affects males; hemophilia A: 1 in 5,617 male births; hemophilia B: 1 in 19,283 male births

A fundamental prerequisite for reliable interpretation is the correct and unambiguous representation of variants, which makes variant validation a critical step.

1.2 NGS and Nomenclatures

The increasing recognition of genetic mutations as major contributors to human disease has been closely linked to the development of technologies capable of detecting sequence variation at large scale. Before the advent of Next-Generation Sequencing (NGS), the study of Mendelian disorders and other inherited traits primarily relied on approaches such as linkage analysis, which aimed to identify chromosomal regions co-segregating with disease within affected families. Genetic

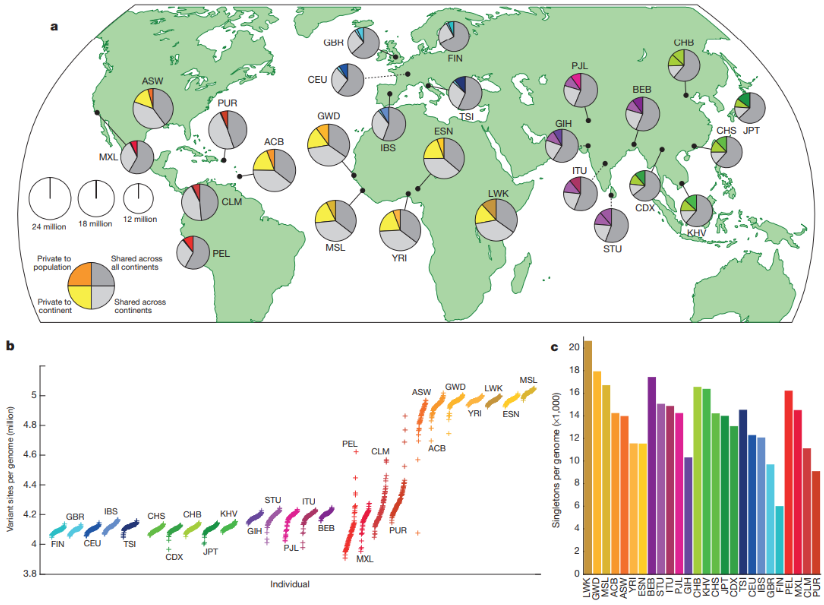

Figure 1.2: **Global distribution of human genetic variation across populations.** The figure summarizes the number and sharing of polymorphic variants identified in different populations, highlighting both population-specific and globally shared genetic variation. Source: [2].

investigations were therefore largely restricted to targeted analyses of candidate genes or broad genomic loci, limiting both resolution and the ability to comprehensively explore genetic variation across individuals and populations [6].

The advent of Next-Generation Sequencing (NGS) technologies has brought to a significantly large increase of explained genetic causes in both rare disease and common but heterogeneous disorders [7]. By enabling the parallel sequencing of millions of DNA fragments, NGS can be applied to analyze large portions of the genome, including whole genomes, exomes, and targeted gene panels. As a result, NGS has become the standard approach for identifying genetic variants in both research and diagnostic [8].

These developments have been profoundly useful in order to minimize the “diagnostic odissey” [9] for patients as whole-genome analysis can be performed in a few days at reasonable costs compared to gene-by-gene analysis based on previous sequencing approaches [10].

Each NGS sequencing experiment typically identifies thousands to millions variants. Even though NGS has led to an unprecedented increase in the volume and diversity of detected genetic variants, the abundance of data generated has shifted the main challenge from variant detection to variant interpretation [11].

In this context, the use of standardized nomenclature systems to refer to variants has become essential [12]. Consistent and unambiguous variant descriptions are required to ensure clear communication among researchers, clinicians, as well as to enable reliable comparison across studies, databases and technologies. Different nomenclature standards have been developed to describe genetic variants at the genomic, transcript and protein levels, thus providing a common language for their representation.

1.3 Motivation and Objectives

While variant detection has become increasingly efficient thanks to NGS technologies, the accurate interpretation and representation of genetic variants remain major challenges. The biological relevance of a sequence alteration depends on its genomic context, its effects at the transcript and protein levels, and the reference systems used to describe it.

Variant validation refers to the computational and methodological process of verifying that a reported genetic variant is syntactically correct, biologically consistent, and unambiguously interpretable with respect to a defined reference system.

Variant validation is therefore a critical step in genomic analysis workflows, as it links raw sequence differences to biologically and clinically meaningful information. However, heterogeneity in reference genomes, transcript models and nomenclature conventions can lead to inconsistencies across validation tools and datasets, thus complicating result comparison and data integration.

The objective of this work is to define and evaluate an independent, efficient and flexible variant validation framework capable of processing heterogeneous input representations, ranging from general genomic descriptions to highly specific transcript and protein level validations. The proposed Variant Validator is designed to operate independently of external web services, ensuring reproducibility, transparency and fast execution.

The presentation of the work is divided into the following chapters:

- Chapter 2 introduces the biological and methodological background required to contextualize genetic variant validation, focusing on the nomenclature of genetic variants, including reference systems and the HGVS nomenclature;
- Chapter 3 describes the methods and implementation details of the proposed variant validation framework. It outlines the software environment, the Variant Validator workflows, resource preprocessing steps, and the integration of rsID-based validation strategies, as well as methodological improvements introduced to address identified limitations;
- Chapter 4 presents the validation and the benchmarking strategy adopted to assess the performance of the Variant Validator framework. It introduces the reference datasets, the mismatch classification criteria, and the comparative analyses to evaluate the baseline implementation and the improved framework;
- Chapter 5 illustrates the application of the proposed Variant Validator tool in a real-world use case. It introduces the VarChat platform as a generative AI-based assistant for genomic variants, describes supported user input modalities, and discusses possible integration strategies and input validation approaches;
- Chapter 6 reports the conclusions of the thesis work and the future developments of the project.

Chapter 2

Background and state of the art

2.1 Biological background

Genetic variants can be classified according to their genomic size, structural characteristics, and predicted functional effects on transcripts and encoded proteins [13]. The main categories of genetic variants are summarized below and are also illustrated in Figure 2.1:

- Single Nucleotide Variants (SNVs) are substitutions involving a single nucleotide. When occurring at a population frequency greater than 1%, they are often referred to as Single Nucleotide Polymorphisms (SNPs) [14];
- Multi-Nucleotide Variants (MNVs) are clusters of two or more nearby nucleotide substitutions occurring on the same haplotype within an individual [15]. Unlike independent single nucleotide variants that happen to be adjacent, MNVs represent a single composite mutational event affecting consecutive bases. When such variants occur within the same codon, their combined effect on the encoded amino acid may differ from the predicted functional consequences obtained by considering each substitution independently. For example, two adjacent nucleotide substitutions may each be individually classified as missense variants, yet together generate a nonsense mutation;
- Insertions and Deletions (INDELs) are short insertions or deletions of one or more nucleotides, less than 50 in length. The number of INDELs in human

genomes is second only to the number of SNPs, and, in terms of base pairs of variation, INDELs cause similar levels of variation as SNPs [16];

- Structural Variants (SVs) are genomic alterations larger than single nucleotide variants (SNVs) but smaller than large-scale chromosomal abnormalities. They are typically defined as rearrangements exceeding 50 base pairs in length and often span from approximately 1 kb to 3 Mb. SVs include deletions, duplications, inversions, translocations, and more complex genomic rearrangements [17];

- Copy Number Variations (CNVs) are segments of DNA that are duplicated or deleted, resulting in variation in the number of copies of a particular genomic region. CNVs account for approximately 5–10% of the human genome, are unevenly distributed-particularly enriched in pericentromeric and subtelomeric regions-and affect different gene groups to varying extents [18].

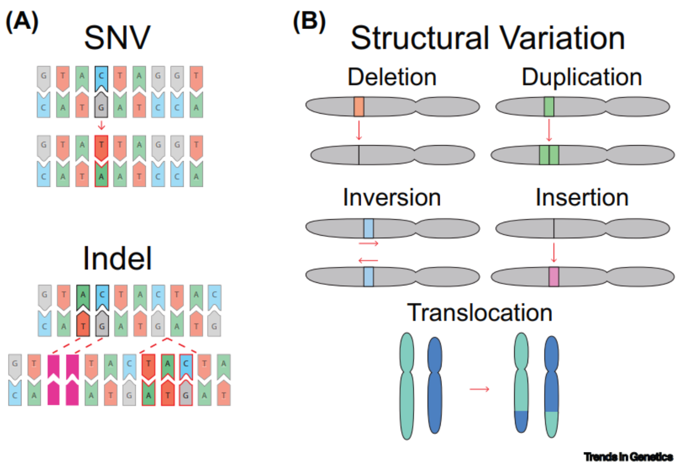

Figure 2.1: **Types of Genetic Variation.** Single nucleotide variants (SNVs) and indels are changes that affect between one and 50 base pairs in a single event. (A) Example of a C>T SNV, and a two base pair deletion and a three base pair insertion indel. Examples of events over $\geq$ 50 base pairs that constitute SVs are shown in (B); these events include deletions, duplications, inversions, insertions, translocations, and complex combinations of these basic variant types. In each example, the top chromosome is the reference, and the variation is highlighted and displayed beneath. Source: [19].

SNP and INDEL variants can also be classified according to their genomic location and functional impact. The principal classifications are shown below:

- Exonic variants: occur within coding exons. They may be:
  - Synonymous (silent), not altering the encoded aminoacid;
  - Missense, causing an aminoacid substitution;
  - Nonsense, introducing a premature stop codon.
- Intronic variants: located within introns. Although often benign, they may affect splicing if positioned near splice donor or acceptor sites, which correspond to the exon–intron junctions recognized at the 5' (donor) and 3' (acceptor) boundaries of introns;
- Splice-site variants: occur at exon-intron boundaries and may disrupt normal RNA splicing;
- Untranslated region (UTR) variants: located in the 5' or 3' UTR and potentially affecting transcript stability or translation efficiency;
- Promoter or regulatory variants: may alter transcription factor binding and gene expression levels;
- Intergenic variants: occur outside annotated genes and may influence distant regulatory elements.

These biological structures form the basis for the representation of genetic variants across different coordinate systems, which are formalized through standardized nomenclature systems such as HGVS.

2.2 Reference Systems and HGVS Nomenclature

2.2.1 Variant Representation: Reference Systems and Formats

In genomic analysis, a reference system provides a standardized coordinate framework against which sequence data and annotations can be interpreted. One of the most common ways of visualizing and organizing genomic information is through annotation tracks, or gene tracks, which display the locations of genomic features relative to a reference sequence. A gene track typically represents the positions of annotated genes or transcripts along a chromosomal coordinate axis, allowing users to examine exon-intron structure, alternative isoforms, and other functional elements in the context of the reference genome [20].

Annotation tracks can include known gene models from curated databases such as RefSeq [21] or Ensembl [22], predicted transcripts, regulatory elements and many other features aligned to the same coordinate system. Genome browsers like the UCSC Genome Browser [23] organize these tracks beneath a common axis of genomic coordinates, enabling rapid visual correlation of different types of information and facilitating interpretation of variant effects in a genomic context.

An example of such visualization is shown in Figure 2.2, where annotated genomic features are displayed along a specific chromosomal region.

A reference genome assembly is a digital representation of an organism DNA sequence, assembled from overlapping sequence data into ordered and oriented contigs and scaffolds that approximate the complete set of chromosomes. For human genomics, assemblies produced by the Genome Reference Consortium (GRC) [24] serve as widely accepted standards. Each assembly provides a unique set of coordinate positions for all bases in the genome, against which sequence reads from an experiment can be aligned and variants can be called.

The most commonly used human reference assemblies are GRCh37, also known as hg19, and GRCh38, also referred to as hg38 [25]. GRCh38 is the more recent build and incorporates improvements in sequence accuracy, representation of alternate haplotypes, and gap closure relative to GRCh37. However, GRCh38 has been slower to gain full adoption, as large amounts of genomic data and annotation tools remain tied to GRCh37 [26]. For this reason, both assemblies continue to coexist in research and diagnostics, even though GRCh38 is widely recognized as the most accurate and updated reference for human genomics.

In addition to genome assemblies and transcript reference systems, standardized databases and data formats play a crucial role in variant identification.

One of the most widely used public repositories of genetic variation is dbSNP (Database of Single Nucleotide Polymorphisms). dbSNP assigns a unique iden-

Figure 2.2: **View of the annotated location of a gene and its features on a genomic RefSeq.** Genomic regions, transcripts and products section of the Full Report. The section for the human RBP4 gene is shown. This is an interactive display of the location of a gene annotated on a genomic RefSeq. More than one genomic RefSeq sequence may be available, and the gene location on any can be displayed using the drop down menu at the top. Zoom and panning functions are enabled, and the tracks displayed can be configured using the "Configure" button. By default, the Genes track displays a merged rendering of transcripts and coding regions. Complete documentation is available by clicking the "?" icon. From NCBI platform. Source: [20].

tifier, known as an rsID (Reference SNP cluster ID), to each submitted variant. For example, a variant may be referred to as rs74315355. An rsID is a stable identifier that links to a record containing genomic coordinates, alleles, population frequencies, and supporting evidence. It allows to refer to a specific variant across resources, even if genomic assemblies or annotations are updates. The genomic position associated with an rsID depends on the reference assembly version, and may differ between GRCh37 and GRCh38.

However, the rsID may be ambiguous, in fact variants at the same base but with different altered alleles are represented with the same rsID. A clarifying example is shown in Figure 2.3.

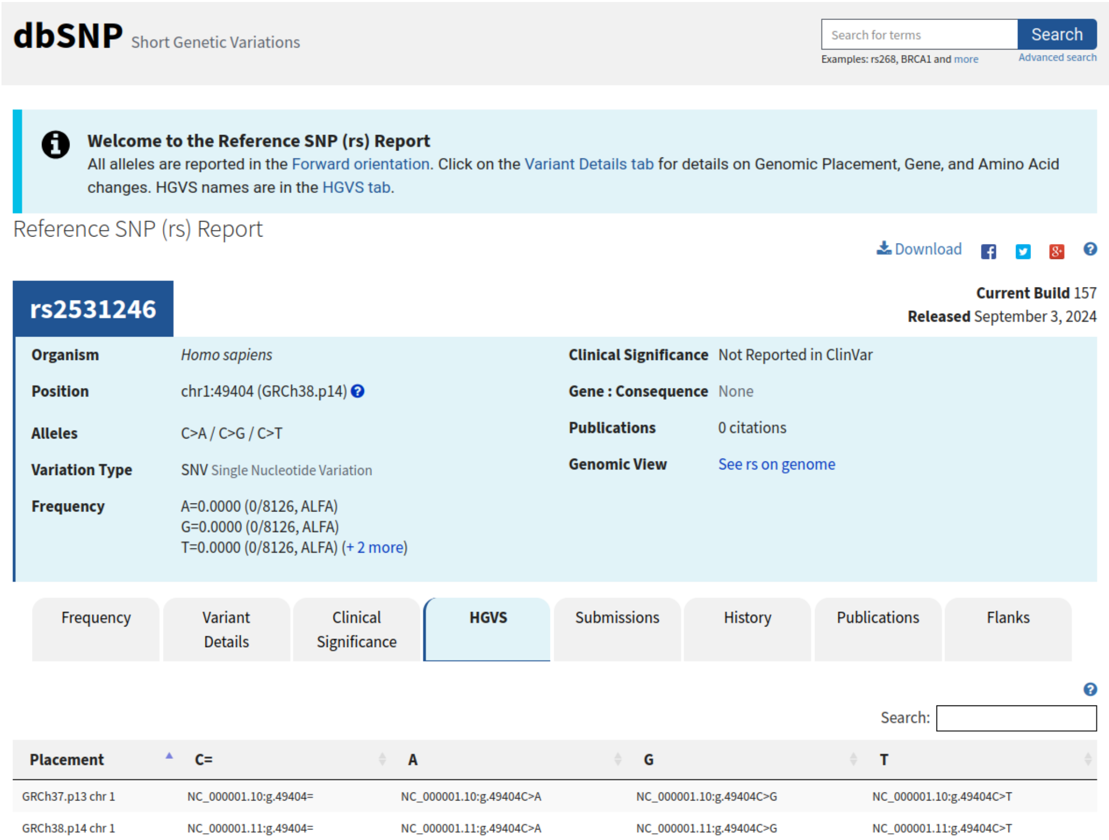

Figure 2.3: **Example of a variant representation on dbSNP.** A single rsID may correspond to multiple alternate alleles at the same genomic position (e.g., C>A, C>G, and C>T), illustrating the potential ambiguity of rsID-based variant identification.

While databases such as dbSNP provide variant identifiers, large-scale sequencing experiments typically store and exchange variants using the Variant Call Format (VCF). VCF is a standardized, tab-delimited text file format designed to represent genetic variants detected in one or more samples in a computationally efficient manner.

Each line of a VCF file corresponds to a single variant and contains mandatory fields that uniquely define its genomic location and allelic composition. The first five columns include: chromosome (CHROM), genomic position (POS), variant identifier (ID, often an rsID), reference allele (REF), and one or more alternate alleles (ALT, comma-separated in the case of multi-allelic variants). Additional columns may report quality metrics, filtering status, and genotype information for individual samples.

A simplified example of a VCF entry is:

```
1    861808    rs13302982    A    G
```

Here, the variant is located on chromosome 1 at position 861,808 of the reference genome, where the reference base A is replaced by G. The coordinate system used in VCF is strictly genomic and assembly-dependent. Consequently, the same biological variant may be associated with different genomic coordinates depending on whether GRCh37 or GRCh38 is used, making it essential to always specify the reference genome build.

In addition to the core fields defining genomic position and allelic composition, VCF files may include additional information such as sequencing coverage (depth), quality scores, and filtering status, which provide insight into the reliability of the variant call.

In addition to single nucleotide substitutions, VCF coordinates can represent insertions, deletions, duplications, and more complex variants. Some examples include:

- `2    179391818    .    ATGCTA    A`
  Represents a deletion of bases from position 179391819 to 179391823 relative to the reference genome.

- `2    179391925    .    A    ATTTTCTTT`
  Represents an insertion of the sequence `TTTTCTTT` after position 179391925.

- `12    57762714    .    A    AA`
  Represents a tandem duplication of a single nucleotide.

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT
1	977028	.	G	T	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	1167796	.	C	T	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	1168180	.	G	C	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	1168567	.	G	A	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	2160304	.	C	G	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	2235378	.	C	T	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	2235513	.	G	A	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	2237525	.	C	T	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	2237542	.	G	A	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	2237665	.	C	T	.	.	DP=648;AF=0.466049;AO=302;RO=346;	
1	2337967	.	G	GC	.	.	DP=648;AF=0.466049;AO=302;RO=346;	

Figure 2.4: **Example of a VCF file.** Few rows of variants located on chromosome 1 are reported. Only first columns related to nomenclature and genotype are shown

An example of a subset of rows of a VCF file is shown in Figure 2.4.

In addition to coordinate systems and variant exchange formats such as VCF, genomic analyses rely on standardized file formats that store reference sequences and structural annotations. Among the most widely used formats are FASTA for sequence storage and GFF/GTF for genomic feature annotation.

The FASTA format is a simple text-based representation of nucleotide or protein sequences. Each sequence entry begins with a header line introduced by the symbol >, followed by lines containing the raw sequence. In the context of human genomics, FASTA files contain complete chromosome sequences used as reference during alignment.

The header line includes an identifier (e.g., chromosome accession or transcript ID) and optional descriptive metadata. The subsequent lines represent the nucleotide sequence using standard IUPAC characters.

An example of the structure of a FASTA file is shown in Figure 2.5.

FASTA files are essential because they define the exact reference sequence against which variants are described. Any coordinate-based annotation (e.g., VCF or HGVS) is meaningful only relative to a specific FASTA reference build (e.g., GRCh37 or GRCh38).

While FASTA files store raw sequences, structural annotation of genomic fea-

```
>1 dna:chromosome chromosome:GRCh37:1:1:249250621:1
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

Figure 2.5: **Example of a FASTA file structure.** The header line (preceded by the character `>`) contains the sequence identifier and reference assembly information (in this case GRCh37 chromosome 1). The subsequent lines represent the nucleotide sequence using standard IUPAC symbols. The presence of consecutive `N` characters indicates unknown bases within the reference genome.

tures is typically provided through GFF (General Feature Format) or GTF (Gene Transfer Format) files. These tab-delimited formats describe the positions of genes, transcripts, exons, coding sequences (CDS), untranslated regions (UTRs), and other functional elements along genomic coordinates.

Each line in a GFF or GTF file corresponds to a genomic feature and includes fields such as chromosome, source, feature type (e.g., gene, exon, CDS), start and end positions, strand orientation, and additional attribute information. The attribute column contains structured key–value pairs specifying identifiers such as gene ID or transcript ID.

GTF is a more strictly defined version of GFF. Both formats allow reconstruction of exon–intron architecture and transcript structure, which is essential for projecting variants between genomic, transcript and protein coordinate systems.

An example of GTF annotation structure is illustrated in Figure 2.6.

2.2.2 HGVS Nomenclature

The Human Genome Variation Society (HGVS) nomenclature provides an internationally recognized standard for the unambiguous description of genetic variants at the DNA, RNA and protein levels [27]. Its primary objective is to ensure consistency, reproducibility, and clarity in the reporting of sequence alterations across laboratories, databases, and clinical settings.

```
1    protein_coding   gene     621059   622053   .
1    protein_coding   transcript   621059   622053
1    protein_coding   exon     621059   622053   .
1    protein_coding   CDS 621099   622034   .    -
1    protein_coding   start_codon 622032   622034
1    protein_coding   stop_codon   621096   621098
1    protein_coding   UTR 622035   622053   .    -
1    protein_coding   UTR 621059   621095   .    -
```

Figure 2.6: **Example of a GTF annotation file.** Each row represents a genomic feature annotated on chromosome 1. The third column indicates the feature type (e.g., gene, transcript, exon, CDS, start_codon, stop_codon, UTR). The fourth and fifth columns specify the start and end genomic coordinates, while the seventh column reports the strand orientation. These structural fields allow reconstruction of transcript architecture and coding regions.

HGVS expressions follow a structured syntax:

`[Reference Sequence]:[prefix].[variant description]`

The reference sequence identifies the exact sequence against which the variant is described, while the prefix specifies the molecular level at which the variant is reported. This structure ensures that both the coordinate system and the affected molecule are clearly defined.

A fundamental requirement of HGVS nomenclature is the use of a well-defined reference sequence. The description of a variant is meaningful only when the underlying reference is explicitly specified, thereby preventing ambiguity. The reference sequence determines the coordinate system used in the variant description and must therefore be explicitly stated. Two main types of reference sequences are recommended:

- LRG (Locus Reference Genomic): stable genomic reference sequences designed for clinical reporting. LRG records are fixed and do not change across genome assembly updates [28];

- RefSeq (Reference Sequence): curated sequences provided by NCBI [29], including genomic (NC_), transcript (NM_), and protein (NP_) accessions.

RefSeq identifiers are versioned and updated as annotations improve.

HGVS variants can be described at different molecular levels, each indicated by a specific prefix:

- *hgvs_g* indicates that the variant is described at the level of genomic DNA, using chromosomal coordinates defined relative to a specific reference genome assembly (e.g., GRCh38 or GRCh37). The position refers to the absolute nucleotide location along the chromosome, independent of any specific transcript.

  For example, in this notation `38:chr17:g.43070945G>T`, `chr17` represents the accession for chromosome 17 (GRCh38 assembly), `g.` specifies genomic-level description, and `43070945G>T` indicates a substitution of guanine with thymine at position 43,070,945 on the chromosome 17;

- *hgvs_c* indicates that the variant is described relative to the coding DNA sequence of a specific transcript. The coordinate system is transcript-dependent and centered on the start codon, where position `c.1` corresponds to the first nucleotide of the initiator codon. Positions upstream of the start codon (5' UTR) are indicated with negative values (e.g., `c.-25A>G`), while positions downstream of the stop codon (3' UTR) are denoted using an asterisk (e.g., `c.*10G>A`). Intronic variants are described relative to the nearest exon boundary using a plus or minus notation. The symbol `+N` indicates a position N nucleotides into the intron downstream of an exon, whereas `-N` indicates a position N nucleotides upstream of an exon within the preceding intron.

  For example, `c.594+1G>A` describes a substitution occurring one nucleotide into the intron following coding position 594, while `c.594-2A>C` indicates a substitution two nucleotides upstream of coding position 594, within the intron preceding that exon;

- *hgvs_p* indicates that the variant is described at the protein level, reflecting the predicted consequence of a DNA or RNA alteration on the amino acid sequence. Protein changes are expressed using the three-letter amino acid code, unless otherwise specified.

For example, in this notation: `NP_009225.1:p.Gln1756Ter`, `NP_009225.1` identifies the protein sequence, `p.` specifies protein-level description, and `Gln1756Ter` indicates that glutamine at position 1756 is replaced by a termination codon (nonsense mutation). Protein-level descriptions may also include missense substitutions (e.g., `p.Gly12Asp`), frameshifts (e.g., `p.Val600Glufs*4`), or in-frame deletions and insertions;

- *hgvs_m* is used for variants occurring in the mitochondrial chromosome. The coordinate system refers to the mitochondrial reference sequence (typically NC_012920).

  For example, in this case: `NC_012920.1:m.3243A>G`, `m.` specifies mitochondrial DNA level, and `3243A>G` indicates a substitution at position 3243 of the mitochondrial genome. Mitochondrial variants are particularly relevant in metabolic and maternally inherited disorders [30];

- *hgvs_n* is used when describing variants relative to a non-coding transcript that does not contain a coding sequence (CDS). Unlike hgvs_c, numbering is based on the full length of the transcript and does not refer to a start codon.

  For example, here: `NR_026971.1:n.152G>A`, `NR_026971.1` denotes a RefSeq non-coding RNA transcript, and `n.152G>A` indicates a substitution at nucleotide position 152 of the RNA sequence. This notation is commonly used for variants affecting long non-coding RNAs (lncRNAs) or other regulatory transcripts [31].

The choice of prefix determines the coordinate system and interpretation of the variant. Each prefix corresponds to a specific reference sequence and numbering scheme, defining how positions are calculated and reported. A summary of the main HGVS prefixes and their associated reference sequence frameworks is provided in Table 2.1.

One of the key principles of HGVS nomenclature is the 3' rule, which resolves ambiguities arising in repetitive sequences. When a variant can be described in multiple equivalent ways, HGVS requires that the alteration be assigned to the most 3' possible position within the reference sequence.

Table 2.1: HGVS reference sequences and their corresponding numbering schemes. Adapted from: [32].

Numbering scheme	**Prefix**	**Position numbering in relation to**
Genomic DNA	`g.`	First nucleotide of the genomic reference sequence
Coding DNA	`c.`	First nucleotide of the translation start codon of the coding DNA reference sequence
Noncoding DNA	`n.`	First nucleotide of the noncoding DNA reference sequence
Mitochondrial DNA	`m.`	First nucleotide of the mitochondrial DNA reference sequence
RNA	`r.`	First nucleotide of the RNA reference sequence
Protein	`p.`	First amino acid of the protein sequence

The example shown in Figure 2.7 illustrates this principle. In the reference sequence, the codon `TTT` encodes phenylalanine (Phe). In the sample sequence, this codon is absent, indicating the deletion of Phe. However, since the surrounding nucleotides contain repetitive elements, the deletion can be represented in two different but biologically equivalent ways: either as `c.1520_1522delTCT` or as `c.1521_1523delCTT`. Both descriptions result in the removal of the same amino acid residue, yet they correspond to slightly shifted nucleotide coordinates.

According to the HGVS 3' rule, the correct representation is the one that places the deletion at the most 3' position possible within the reference sequence. Therefore, the preferred description is `c.1521_1523delCTT`. This convention ensures a unique and standardized annotation, preventing multiple alternative descriptions of the same molecular event.

Due to the strict syntax and multiple possible representations of the same biological variant, automated validation of HGVS expressions is essential to ensure correctness and consistency.

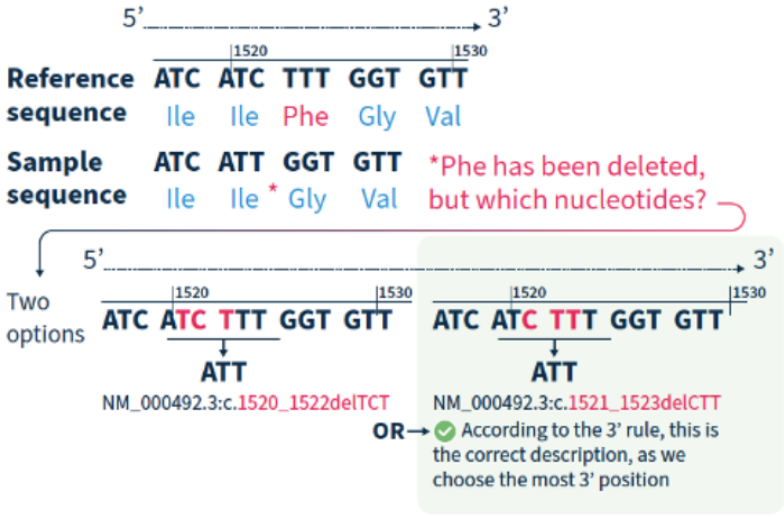

Figure 2.7: **Illustration of the HGVS 3' rule.** Example of an ambiguous codon deletion within a repetitive sequence. Although the phenylalanine codon (TTT) is removed in the sample sequence, the deletion can be represented by two equivalent nucleotide descriptions due to sequence repetition. According to the HGVS 3' rule, the correct annotation is the one assigned to the most 3' position in the reference sequence.

2.3 State of the art in variants validation tools

To address the objectives defined in this work, a state of the art of the existing variants validation tools was performed. Each tool was evaluated with respect to its functional scope, input flexibility, architectural design and execution environment.

2.3.1 Overview of variants validation tools

LOVD Syntax Checker

The LOVD HGVS library (HGVS-syntax-checker) [33] is an open-source, standalone PHP validation engine aimed at checking variant descriptions according to HGVS nomenclature rules. It is available at https://github.com/LOVDnl/HGVS-syntax-checker. Unlike sequence-level validators, it does not require access to a reference genome or transcript sequence; instead, it detects syntax and formatting errors, corrects common user mistakes, and outputs standardized HGVS representations. For full sequence-aware validation and biological consistency checks, external tools are required.

The library can be executed locally (PHP CLI) or embedded as a PHP dependency, making it fully usable without remote services. An online interface and a lightweight API are also available for low-volume requests. An example of its behavior is shown in Figure 2.8.

When the variant `NM_000518.4:c.20delA` was provided as input, the library flagged the deleted nucleotide as redundant and suggested the normalized HGVS-compliant form `NM_000518.4:c.20del`. This illustrates the tool's role as a syntax-aware normalizer, enforcing HGVS formatting rules without performing sequence-level validation.

```
](base) engenome@engenome:~/PycharmProjects/HGVS-syntax-checker$ php -f HGVS.php "NM_000518.4:c.20delA"
[
    {
        "input": "NM_000518.4:c.20delA",
        "identified_as": "full_variant_DNA",
        "identified_as_formatted": "full variant (DNA)",
        "valid": false,
        "messages": [],
        "warnings": {
            "WSUFFIXGIVEN": "The deleted sequence is redundant and should be removed."
        },
        "errors": [],
        "data": {
            "position_start": 20,
            "position_end": 20,
            "position_start_intron": 0,
            "position_end_intron": 0,
            "range": false,
            "type": "del"
        },
        "corrected_values": {
            "NM_000518.4:c.20del": 1
        }
    }
```

Figure 2.8: **Example of output of the LOVD HGVS syntax checker.** For the input `NM_000518.4:c.20delA`, the tool identifies the deleted nucleotide as redundant and suggests the normalized HGVS-compliant representation `NM_000518.4:c.20del`.

Mutalyzer

Mutalyzer 2 [34] is an open-source HGVS nomenclature checker designed to help users produce unambiguous, HGVS-compliant variant descriptions. The core tool is the Name Checker, which takes an interpretable HGVS description and returns a canonical one by applying a fixed pipeline: it first performs a syntactic check (parsing the description with a formal grammar), then a semantic check (verifying that coordinates and reference nucleotides match the underlying sequence), and finally a disambiguation step (e.g., minimization, simplification of variant types, and $3'$ shifting according to HGVS rules).

A key practical drawback is that Mutalyzer's sequence-level validation and coordinate conversions rely on retrieving reference sequences and mapping information from external repositories, primarily NCBI. In practice, this entails API/database queries to NCBI (including additional queries for annotation enrichment), making Mutalyzer not fully self-contained and potentially sensitive to network availability. An example of Mutalyzer usage in a programmatic setting is shown below.

Given the input `NM_000059.3:c.123T>G`, the tool returns a `corrected_description` but also reports a semantic error (`SEQUENCEMISMATCH`), indicating that the specified reference nucleotide (`T`) does not match the base found in the reference sequence at that position.

In the same run, the warning related to NCBI E-utilities highlights that reference retrieval is performed through remote queries, confirming the dependence on API calls to NCBI for sequence-level validation.

VariantValidator

VariantValidator [35] is a web-based tool designed to validate, map, and format sequence variant descriptions according to HGVS recommendations. It was developed to make the functionality of the open-source Biocommons `hgvs` Python package [36] accessible through a user-friendly interface, while adding practical features that are commonly needed in clinical reporting and when handling high-throughput variant calls.

VariantValidator validates variants in the context of the underlying reference sequence, thus detecting sequence mismatches. In addition, VariantValidator supports the conversion between HGVS descriptions and VCF representations, and it can map chromosomal variants to the set of relevant transcripts overlapping the same genomic locus, presenting the results in a single consolidated view.

At the computational core, these operations rely on the Biocommons `hgvs` package, whose architecture is summarized in Figure 2.9. The workflow starts from a parser that converts an HGVS string into a structured SequenceVariant object (encoding accession, coordinate system and the position/edit components). The same object can then be processed by modules that validate internal consistency, and validate externally against reference sequences and transcript models. Downstream, a normalizer rewrites equivalent descriptions into a canonical HGVS form (e.g., by trimming shared sequence context and applying HGVS shifting rules), while a mapper projects variants between coordinate systems using transcript-genome alignments.

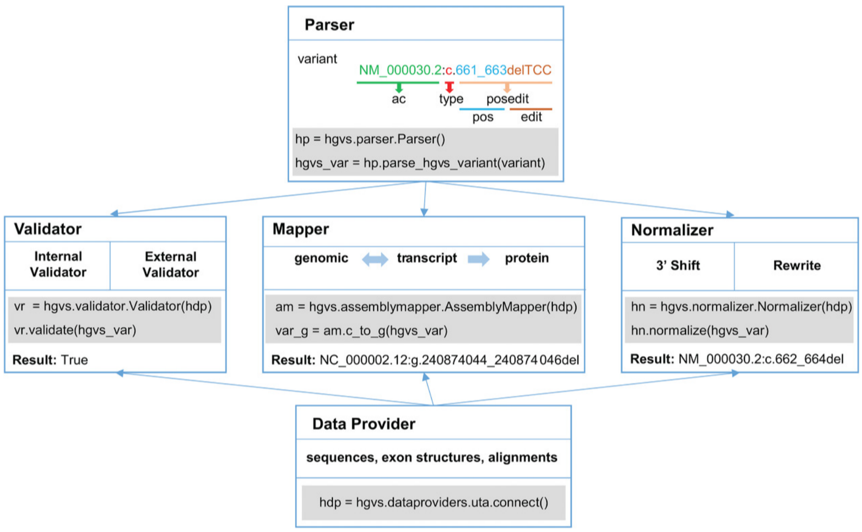

Figure 2.9: **Overview of the Biocommons `hgvs` package architecture.** The diagram summarizes the main components of the library: an HGVS string is parsed into a structured `SequenceVariant` object (Parser), then checked for internal and sequence-aware consistency (Validator), optionally rewritten into a canonical HGVS representation (Normalizer), and projected across coordinate systems (Mapper) using reference data and transcript-genome alignments. Source: [36]

VariantValidator is implemented around a local reference infrastructure. It relies on a local installation of UTA (Universal Transcript Archive), a relational database that stores transcript–genome alignments, exon structures, and coordinate mappings, and on a local SeqRepo instance for sequence retrieval. UTA provides the necessary alignment framework to project variants between genomic and transcript coordinate systems, while SeqRepo enables efficient access to reference genomic, transcript, and protein sequences. These resources are complemented by auxiliary lookup tables that are periodically synchronized with external providers such as NCBI, HGNC, and UCSC.

When focusing on the underlying `hgvs` package as a building block, an important practical limitation is that while `hgvs` supports forward projections such as c→g and c→p, it does not provide an inverse mapping from protein-level descriptions back to transcript coordinates (p→c). As a result, cross-level ambiguity resolution is not inherently solved inside the core library and must be addressed externally, which becomes a relevant limitation when compared with tools that attempt to manage these ambiguities more directly.

SnpEff

SnpEff [37] is an open-source tool designed for the functional annotation of large variant datasets, typically produced by NGS pipelines. Unlike HGVS-focused validators such as Mutalyzer or VariantValidator, which primarily aim to check and normalize a single variant description against a specific reference sequence, SnpEff is mainly used downstream of variant calling to assign each variant a genomic context and a predicted biological effect.

Given a set of variants (most commonly provided in VCF format), SnpEff classifies each event according to the genomic region it overlaps-for example intronic, exonic, untranslated regions (5′UTR/3′UTR), upstream/downstream of transcripts, splice-site, or intergenic regions. When the variant affects a protein-coding transcript, the tool also predicts the coding consequence, distinguishing effects such as synonymous vs. non-synonymous substitutions, frameshifts, stop-gain/stop-loss events, and start-gain/start-loss events. If multiple transcript isoforms are annotated for the same gene, SnpEff reports consequences for each transcript, since the predicted effect can differ depending on the isoform considered.

Operationally, SnpEff works in two stages. First, it builds a genome-specific database starting from a reference genome sequence (FASTA) and an annotation file (e.g. GTF/GFF). In the annotation stage, the database is loaded and variants are rapidly intersected with genomic intervals using efficient interval-based data structures (an “interval forest” indexed by chromosome), enabling high-throughput annotation at scale. The output can be generated again as a VCF, with predicted effects stored in the INFO field, which makes SnpEff easy to integrate into automated pipelines and batch analyses.

Unlike tools such as Mutalyzer or VariantValidator, SnpEff is not an HGVS validator and does not parse or validate c. or p. descriptions provided as input. Instead, it accepts variants in VCF format, meaning that the input must be defined only at the genomic level using chromosome coordinates, reference allele, and alternate allele relative to a specific genome assembly.

TransVar

TransVar [38] is an open-source variant validation tool specifically designed to resolve ambiguities that arise when translating variants across genomic (g.), transcript (c.), and protein (p.) coordinate systems.

In the human genome, relationships between variant representations are rarely one-to-one. A single genomic variant may affect multiple transcripts (one-to-many), while a protein-level variant may originate from distinct genomic alterations (many-to-one). In complex loci, these relationships may become many-to-many. Ambiguity becomes particularly problematic when transcript identifiers or isoform information are missing. For example, the protein variant `EGFR:p.L747S` can derive from different genomic substitutions across distinct isoforms. Without specifying the transcript context, its genomic origin cannot be uniquely determined.

TransVar was developed to explicitly expose and systematize these cross-level ambiguities. It enables complete cross-level mapping among g., c., and p. coordinates: a variant provided at any level can be translated to the other two while systematically enumerating all compatible isoforms and genomic candidates. This multi-level mapping framework is illustrated in Figure 2.10.

TransVar provides three main functionalities:

- Forward annotation: given a genomic variant (g.), TransVar reports all compatible transcript (c.) and protein (p.) consequences across all isoforms in the selected transcript database;

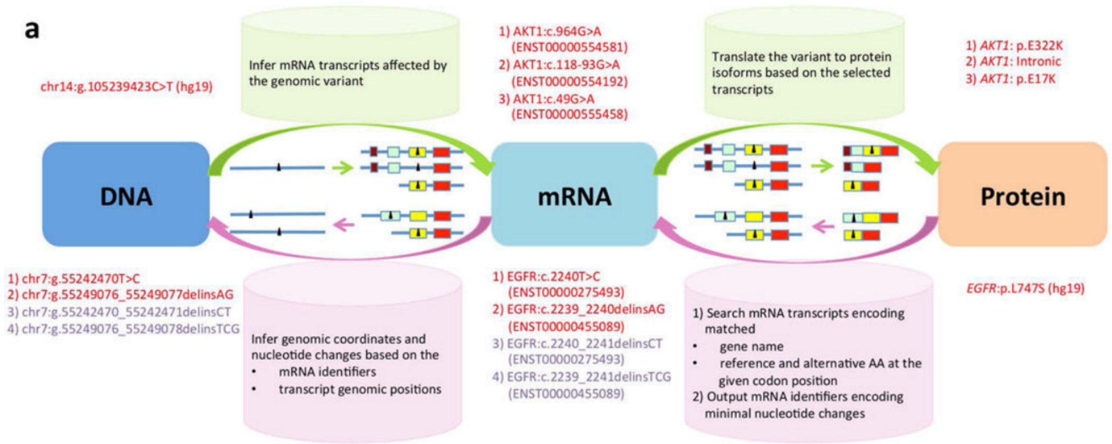

Figure 2.10: **TransVar cross-level mapping framework.** The diagram illustrates complete translation across genomic (DNA), transcript (mRNA), and protein levels. A genomic variant may affect multiple transcripts and generate different protein consequences, while a protein-level description may correspond to multiple genomic candidates depending on the isoform context. Source: [38].

- Reverse annotation: given a transcript-level (c.) or protein-level (p.) variant, TransVar returns all possible genomic origins capable of producing that description, thereby revealing whether the variant is uniquely determined or intrinsically ambiguous;
- Equivalence annotation: given a protein variant, TransVar identifies equivalent protein descriptions generated by the same genomic alteration but annotated on different isoforms.

TransVar supports multiple genome builds (hg19 and hg38), with build switching available via command-line options. It accepts input either in HGVS format or tabular form, and supports batch processing.

Genomic variants can also be provided in standard VCF format, making it possible to annotate variants derived directly from NGS pipelines. When VCF input is supplied, TransVar projects each variant onto all compatible genes and transcripts and computes the corresponding functional consequences onto the other levels.

Several transcript databases can be configured, including RefSeq, Ensembl, GENCODE [39], and UCSC. The choice of database directly influences annotation breadth and interpretation. RefSeq provides a conservative and highly curated transcript set, typically focusing on well-supported isoforms. In contrast, Ensembl and GENCODE include a broader and more heterogeneous repertoire of alternative and computationally predicted transcripts. As a consequence, using Ensembl or GENCODE may increase isoform coverage and the number of reported variant consequences.

A key strength of TransVar is its isoform-aware design. Unlike many validation tools that report only a single representative transcript, TransVar systematically annotates all compatible isoforms, preventing information loss and making structural ambiguity explicit.

TransVar follows HGVS normalization principles. The $3'$ rule is applied at the nucleotide level (genomic and transcript coordinates). As discussed in Par. 2.2.2, when multiple equivalent representations are possible-typically in repetitive sequences-the variant must be assigned to the most $3'$ position within the reference sequence. The tool reports both normalized and alternative aligned representations, increasing transparency across coordinate systems.

Ambiguity at the protein level is handled explicitly by TransVar. Generally, multiple codons may encode the same amino acid due to the degeneracy of the genetic code. TransVar first identifies the reference codon at the specified protein position and then enumerates all possible nucleotide substitutions-including both SNVs and MNVs-that would produce the altered amino acid. For each candidate, the number of nucleotide changes relative to the reference codon is computed, and the representation requiring the smallest number of substitutions is considered the most parsimonious solution. At the same time, alternative valid genomic representations are also reported. In this way, the inherent ambiguity introduced by codon degeneracy is made explicit rather than being resolved implicitly by arbitrarily selecting a single candidate.

An example of this behavior is shown in Figure 2.11, where multiple genomic candidates are reported for a single protein-level description.

The output includes variant class, transcript context, exon number, and genomic, transcript, and protein coordinates when applicable.

Additional features include reporting of candidate MNVs, integration with dbSNP for rsID reporting (for single-nucleotide variants), and transcript catalog ex-

```
PIK3CA:p.E545K       ENST00000263967 (protein_coding)       PIK3CA  +
   chr3:g.178936091G>A/c.1633G>A/p.E545K     inside_[cds_in_exon_10]
   CSQN=Missense;reference_codon=GAG;candidate_codons=AAG,AAA;candidate_mnv_vari
   ants=chr3:g.178936091_178936093delGAGinsAAA;dbsnp=rs104886003(chr3:178936091G
   >A);aliases=ENSP00000263967;source=Ensembl
```

Figure 2.11: **Protein-to-genome mapping example in TransVar.** Output of the command line `transvar panno -i PIK3CA:p.E545K --ensembl`. The tool maps the protein variant to genomic and transcript coordinates, reports the reference codon (GAG), enumerates candidate codons producing the amino acid substitution (AAG, AAA), lists alternative genomic representations including MNVs, and provides exon context and known dbSNP identifiers. [40]

traction for a given gene. However, TransVar does not accept rsID identifiers directly as input; variants must be provided in coordinate or HGVS form.

TransVar occupies a distinct position within the landscape of variant validation tools. Unlike SnpEff, which primarily operates on VCF input and focuses on genomic effect prediction, TransVar supports direct input at g. (in both VCF-like or HGVS formats), c., and p. levels and performs exhaustive cross-level translation. Its primary strength lies in systematically resolving isoform-related ambiguity and enumerating all equivalent representations across coordinate systems. This makes it particularly suitable for studies requiring comprehensive multi-level annotation and ambiguity-aware variant interpretation.

Ensembl Variant Effect Predictor (VEP)

The Ensembl Variant Effect Predictor (VEP) is a widely adopted variant annotation tool developed by the Ensembl project [41]. It is designed to predict the molecular consequences of genomic variants and to integrate functional, population, and clinical annotations within a single framework.

VEP supports both VCF-like and HGVS inputs (including transcript- and protein-level descriptions), but it is primarily optimized for high-throughput annotation of VCF files.

Each variant is compared against a selected transcript set (Ensembl/GENCODE or RefSeq), and consequences are assigned using Sequence Ontology (SO) terms [42]. For every variant-transcript pair, VEP reports genomic, cDNA, and protein coordinates, codon and amino acid changes, exon context, transcript biotype, and predicted impact severity.

A major strength of VEP is its integration of external resources, including:

- population allele frequencies (e.g., gnomAD [43], 1000 Genomes [44]);
- clinical databases (ClinVar [45]);
- somatic mutation resources (COSMIC) [46];
- protein domain annotations (InterPro) [47];
- pathogenicity prediction scores (SIFT [48], PolyPhen-2 [49], CADD [50], SpliceAI [51]);
- regulatory region annotations.

This makes VEP particularly suitable for large-scale variant prioritization and clinical interpretation workflows.

VEP is accessible through three main interfaces: (i) a web-based graphical interface, (ii) a REST API for programmatic access, and (iii) a command-line tool for local execution. The web interface is convenient for small datasets, whereas large-scale analyses typically require either API-based workflows or local installation.

Although VEP supports HGVS input (including transcript- and protein-level descriptions) and performs coordinate translation across genomic (g.), transcript (c.), and protein (p.) levels similarly to TransVar, this functionality is only delivered through the Ensembl web interface or via REST API services.

While a command-line version of VEP is available, full local deployment is not available. This highlights that HGVS-based input parsing and cross-level coordinate translation can be performed only by online Ensembl services or by an API-backed infrastructure.

As a consequence, VEP is not inherently designed as a fully self-contained validation engine. In contrast, tools such as TransVar are conceived as autonomous local systems, where cross-level mapping (g./c./p.) can be performed without dependence on external services.

2.3.2 Functional and technical requirements

The analysis of existing tools highlights a central challenge in variant validation: the need for a fully automated system capable of performing accurate multi-level coordinate translation while minimizing cross-level ambiguity.

To summarize the comparative analysis of the reviewed tools, Table 2.2 provides an overview of their main functional and architectural characteristics. The comparison focuses on deployability, input flexibility, cross-level mapping capabilities, ambiguity handling, and dependency on external services.

From a functional perspective, the desired Variant Validator must:

- support input at all relevant representation levels: genomic (g.), coding transcript (c.), protein (p.), mitochondrial (m.), non-coding transcript (n.), rsID and VCF-like formats;
- perform consistent cross-level mapping (g $\leftrightarrow$ c $\leftrightarrow$ p) while preserving HGVS compliance;
- explicitly expose ambiguity arising from isoform multiplicity rather than masking it through default transcript prioritization;
- enumerate equivalent variant representations when multiple genomic events can produce the same transcript or protein-level change;
- correctly normalize variants according to HGVS alignment rules at each coordinate level.

From a technical perspective, the system must:

- operate as a fully self-contained local tool without reliance on web services or REST APIs;
- be executable in controlled or offline computational environments;
- allow configurable transcript databases while maintaining reproducibility;
- support batch processing for integration into automated pipelines.

The review of current tools shows that many platforms either prioritize effect prediction over cross-coordinate validation (e.g., SnpEff), focus primarily on HGVS syntax validation (e.g., LOVD Syntax Checker), or require web/API infrastructure to enable full multi-level mapping (e.g., Ensembl VEP and Mutalyzer).

In contrast, TransVar provides native support for genomic, transcript, and protein inputs; performs comprehensive cross-level mapping locally; systematically enumerates isoform-dependent consequences; and makes ambiguity explicit rather than implicit. Its architecture aligns most closely with the functional and technical requirements identified above.

For these reasons, TransVar was selected as the foundational Variant Validation engine for the development of the proposed system.

Table 2.2: Comparative overview of variant validation tools analyzed in this work.

Tool	**Locally deployable**	**External services independent**	**HGVS parsing**	**g↔c↔p mapping**	**Explicit ambiguity handling**
LOVD Syntax Checker	Yes	Yes	Yes (syntax only)	No	No
Mutalyzer	No	No	Yes	Yes	No
VariantValidator	Yes	Yes	Yes	No	No
SnpEff	Yes	Yes	No	No	No
TransVar	Yes	Yes	Yes	Yes	Yes
VEP	No	No	Yes	Yes	Yes

Chapter 3

Methods

In this chapter, the methodological framework developed for the implementation of the Variant Validation module is presented. The chapter describes the computational environment, the Variant Validation workflow built upon the TransVar tool, and the set of conversion strategies designed to map across different levels of genomic annotation, including genomic coordinates, coding DNA (cDNA) coordinates, and protein-level representations.

3.1 Software and Computational Environment

TransVar is the variant validation engine selected as the foundation of this work. TransVar is implemented in Python and distributed as an open-source command-line tool [38]. The latest official release of TransVar was published on July 1, 2018.

Python was therefore used as the programming language for the methodological improvements and for the development of the entire variant validation workflow. Python has well-established applications in bioinformatics, particularly in the analysis and processing of DNA sequences. As an interpreted language supporting both procedural and object-oriented paradigms, it combines flexibility with readability. Furthermore, Python provides a large ecosystem of libraries facilitating file handling.

Code development and modifications were performed using PyCharm, an Integrated Development Environment (IDE).

Version control was managed through Bitbucket, a web-based Git repository management platform developed by Atlassian. Bitbucket supports distributed version control, structured branching strategies, and traceability of code changes, ensuring controlled evolution of the codebase and reproducibility of experimental configurations.

To guarantee environmental consistency and portability, the execution environment was containerized using Docker. Docker employs operating system-level virtualization to package applications and their dependencies into containers thus allowing the system to be executed in a reproducible and deterministic manner across heterogeneous infrastructures [52]. This approach minimizes discrepancies caused by system-level configuration differences and aligns with the objective of maintaining a fully self-contained validation framework.

Cloud-based components were deployed within the Amazon Web Services (AWS) ecosystem. AWS is a comprehensive cloud computing platform that provides scalable infrastructure services, including compute resources, storage solutions, and managed database systems. Within this environment, AWS DocumentDB was adopted as a managed document-oriented database service compatible with MongoDB APIs [53]. DocumentDB was specifically integrated to support TransVar rsID-based input handling by storing and querying this type of variant records. Thus this architectural layer enables cross-level mapping of rsID inputs. The detailed design and operational logic of the rsID integration module are described in Par. 3.2.4.

3.2 Variant Validator framework

3.2.1 Resources preprocessing

Before performing variant validation, TransVar transforms heterogeneous genomic resources into a structured and queryable internal representation. This preprocessing step acts as a normalization layer between raw biological annotation files and the variant validation engine. Importantly, this transformation is performed once during database preparation and not at each validation request. Once generated, the internal database can be reused across multiple variant queries, ensuring consistent behavior and computational efficiency.

The preprocessing workflow relies on three categories of resources, all discussed in Par. 2.2.1:

- a reference genome sequence (FASTA format);
- transcript and gene annotation files (GTF/GFF formats from RefSeq, Ensembl, GENCODE, UCSC);
- optional external variant resources (e.g., dbSNP).

Because transcript annotations are distributed as large, hierarchical GTF/GFF files, directly interrogating them at runtime would require repeated parsing, interval scanning, and on-the-fly reconstruction of gene-transcript structures, resulting in unnecessary long computational effort and non-deterministic behavior across heterogeneous sources. For this reason, preprocessing converts the raw annotation files into a compact, transcript-centric internal database in which each transcript is represented in a standardized and self-contained form and can be retrieved through dedicated indices. This design enables fast and reproducible transcript lookup during variant validation.

Since the preprocessing procedure is independent from individual validation requests, raw resources can be updated whenever new genome assemblies, transcript releases, or annotation revisions become available. Rebuilding the internal database with updated reference files allows the Variant Validator to remain aligned with evolving biological knowledge while preserving the same validation logic.

Transcript annotation files are parsed and transformed into structured gene-transcript models. For each transcript, genomic span, strand orientation, exon structure, CDS boundaries, transcript identifiers (with version), and cross-references are extracted and harmonized. The result is a standardized internal representation independent of the original annotation provider. Normalized transcripts are serialized into a compact internal database file with extension `.transvardb`. Each line represents a fully reconstructed transcript model, collapsing the hierarchical structure of the original GFF/GTF files into a single transcript-level record.

An example entry derived from the GRCh38 RefSeq GFF annotation for a specific transcript of the gene *BRCA1* is shown in Table 3.1. Each row corresponds to a structured field stored in the internal `.transvardb` database.

Field	**Value**
Gene symbol	BRCA1
Transcript ID	NM_007294
Transcript version	4
Transcript biotype	protein_coding
Transcript span	43044295–43125364
Chromosome	chr17
Strand	-
CDS boundaries	43045678–43124096
Exon coordinates	[(43044295,43045802), ... , (43125271,43125364)]
Protein accession	NP_009225
External identifiers	GeneID:672, HGNC:1100, MIM:113705

Table 3.1: Example of transcript-centric record stored in the internal `.transvardb` database for the gene *BRCA1*.

The transformation from multi-line relational annotation (gene → transcript → exon → CDS) into a transcript-centric structure is fundamental to enabling deterministic variant mapping. After transcript models are serialized into the `.transvardb` file, additional indexing structures are created to enable efficient and fast querying during variant validation.

Two complementary access mechanisms are implemented. The first is a *name-based index*, which allows direct retrieval of transcript records using gene symbols or transcript identifiers. This mechanism is essential when the input variant is expressed in transcript-level notation (e.g., HGVS c. or HGVS p.), where the reference transcript must be resolved before coordinate mapping can occur.

Operationally, the name-based access layer is implemented through two indices defined as in-memory hash-based lookup structures: a gene index, which maps each gene symbol to the starting position of its associated transcript records in the `.transvardb` file, and a transcript index, which maps each transcript identifier to its corresponding file offset. The transcript index is used for transcript-level input, enabling direct retrieval of a specific transcript record, whereas the gene index is used for gene-level input to locate the block of transcripts associated with the given gene, which are then retrieved sequentially. These positional offsets allow the validator to seek directly to the relevant records in the database file, avoiding full scans of the complete transcript collection.

The second is a *coordinate-based index*, which organizes transcripts according to their genomic positions. This structure enables rapid identification of all transcripts overlapping a given genomic interval, a fundamental requirement when the input variant is provided in genomic notation (e.g., genomic coordinates).

At the implementation level, the coordinate-based index is implemented as a separate genomic interval file derived from transcript coordinates, sorted by chromosome and genomic position, compressed with `bgzip`, and indexed with `tabix`. Given a genomic query interval, this index can be interrogated to retrieve only the subset of transcript identifiers whose genomic span intersects the queried locus. These identifiers are then resolved through the name-based indexing layer to access the corresponding transcript records in the `.transvardb` database.

The use of indexing enables fast retrieval of relevant transcript records, allowing variant validation to be performed efficiently without scanning the entire `.transvardb` database.

In addition to transcript annotations, external variant databases such as dbSNP can be configured as supplementary resources. Unlike transcript annotation files, dbSNP entries are not reconstructed into the internal `.transvardb` format.

Instead, the dbSNP dataset, typically provided as a coordinate-indexed VCF file, is accessed directly through its existing genomic indexing structure (e.g., tabix indexing).

At runtime, when a variant is being validated, its genomic coordinates are used to query the dbSNP resource within the relevant interval. If a matching record is found, based on chromosome, position, and allele information, the corresponding dbSNP identifiers are appended to the validation output.

Importantly, dbSNP does not influence transcript reconstruction, coordinate mapping, or functional consequence inference. It functions exclusively as an external lookup layer that enriches the final validation by reporting whether the queried variant corresponds to a previously catalogued polymorphism.

The overall preprocessing flow is shown in Figure 3.1.

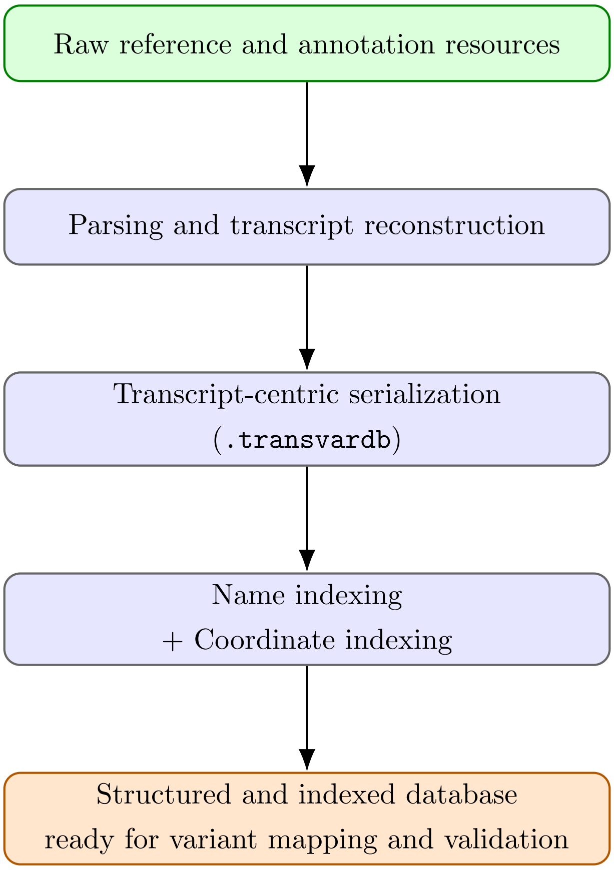

Figure 3.1: **Preprocessing workflow for transcript database construction.** The diagram illustrates the sequential transformation from raw reference and annotation resources to a structured and indexed internal database ready for variant mapping and validation.

3.2.2 Variant Validator workflows

After the preprocessing step has established a structured and indexed transcript database, the validation engine processes input variants according to their representation level. In this section, the three possible input workflows, genomic (g.), coding DNA (c.), and protein (p.), are detailed.

An important aspect of these processes is the definition and application of appropriate normalization rules to all three level inputs, ensuring that variant representations are corrected and standardized in accordance with the HGVS nomenclature guidelines.

3.2.2.1 Genomic input workflow (g.)

Genomic variants (`g.`) represent alterations described with respect to reference genome coordinates, where positions are defined on a specific genome assembly (e.g., GRCh37 or GRCh38). Variants expressed at this level are processed through the TransVar `ganno` command-line, which activates the genomic-driven validation workflow.

After the preprocessing stage has established a normalized and indexed internal transcript database, variant validation operates on individual or batch queries. In batch mode, multiple variant entries are provided as a list or input file and processed in parallel within a single execution, enabling efficient high-throughput validation. When a variant is provided at the genomic level,either as an HGVS-like `g.` expression or as a VCF entry, the system follows a structured workflow that transforms a coordinate-based description into a transcript-aware and functionally interpreted validation.

Input parsing and normalization

A genomic input is first parsed into an internal representation that captures chromosome, coordinates range, and reference/alternate alleles. Because genomic variants may be expressed using slightly different conventions (e.g., chromosome naming schemes such as `chr17` vs `17`), chromosome identifiers are harmonized to match the selected reference assembly.

Reference validation is then performed against the chosen genome assembly. The reference genome FASTA file is accessed through an indexed interface (FASTA index, e.g., `faidx`), which enables direct fast retrieval of the nucleotide sequence corresponding to the specified genomic coordinates without scanning the entire file. The nucleotide sequence obtained from the assembly is compared to the reference allele provided in the input.

If the supplied reference allele does not match the assembly sequence at the specified position or interval, the variant is flagged as inconsistent with the selected genome build. Such discrepancies may arise, for example, when coordinates refer to a different reference assembly (e.g., GRCh37 vs GRCh38) or when the variant description is incorrectly specified.

By validating the reference allele through indexed sequence retrieval, the workflow ensures that all downstream coordinate projections are anchored to the exact nucleotide content of the chosen genome assembly.

For multi-nucleotide substitutions and insertions or deletions, additional normalization procedures are applied to remove redundant sequence context and to establish a canonical HGVS representation of the event. This step ensures consistency between equivalent representations of the same biological variant.

For example, in repetitive genomic regions, the same insertion or deletion may admit multiple coordinate descriptions. For instance, considering the sequence:

```
A T A T A T
```

and considering that the first A is at the genomic coordinate 100, if one `AT` repeat is deleted, the event could be described as:

```
g.100_101delAT
g.102_103delAT
g.104_105delAT
```

All three descriptions correspond to the removal of a single repeat unit and are biologically indistinguishable. Normalization resolves this ambiguity by selecting a consistent HGVS representation following the $3'$ rule, thus ensuring correct reporting.

Context identification and transcript collection

Once normalized, the genomic coordinates are used to query the indexed transcript database (`.transvardb`), retrieving all transcripts overlapping the queried interval through the coordinate-based indexing strategy described in Par. 3.2.1. Because multiple transcripts, and so potentially multiple genes, may overlap the same genomic locus, a single input variant may correspond to several valid transcript contexts. The workflow therefore branches into parallel validation options, one for each relevant transcript.

This transcript-dependent interpretation is a fundamental property of genomic-level validation: the same genomic variant may lead to different cDNA and protein-level outcomes depending on exon structure, coding boundaries, and strand orientation. For example, a variant may be exonic and missense in one transcript, but intronic or untranslated in another.

This projection from genomic coordinates to coding DNA (`c.`) and, when applicable, protein-level (`p.`) descriptions ensures that the functional interpretation remains consistent with the selected transcript model.

Genomic-to-cDNA mapping

For each overlapping transcript, genomic coordinates are translated into cDNA coordinates using the exon structure and strand information stored in the internal pre-processed database. This step relies on the explicit mapping between genomic positions and the ordered exon segments that define each transcript.

If the variant is within an exon, the corresponding position in the spliced transcript is computed by accounting for exon boundaries and cumulative exon lengths. If the variant is within an intron, its position is represented relative to the nearest exon boundary according to HGVS conventions. When a transcript is located on the negative strand, the mapping procedure additionally reverses coordinate orientation and applies reverse-complement transformation to the nucleotide sequence, ensuring that the resulting cDNA description follows the 5'→3' direction of the transcript.

Illustrative examples of genomic-to-cDNA mapping

To clarify how genomic coordinates are converted into cDNA coordinates, a simplified transcript model located on the positive strand with the following structure is considered:

- the first exon, including a 5' UTR segment and a coding exon, has genomic location 100–149;
- the second exon, including a coding exon and a 3' UTR segment, has genomic location 200-249;

- the coding sequence (CDS), beginning with the Start Codon ATG, starts in the genomic position 110 and includes all the coding exons of the gene (in this example, up to the genomic position 230).

In this configuration, the CDS spans two coding exonic segments (110–149 and 200–230), while the region 150–199 corresponds to an intron.

Example 1: Exonic variant

Supposing a genomic SNV occurs at position `g.205A>G`, because position 205 lies within the CDS portion of Exon 2 (200–230), it is included in the ordered list of coding nucleotides constructed during preprocessing. To compute the cDNA coordinate, the system counts the cumulative number of coding bases preceding the variant:

- 110–149: 40 coding bases (from Exon 1);
- 200–204: 5 coding bases (from Exon 2).

Thus, position 205 corresponds to the 46th nucleotide of the CDS. The resulting transcript-level representation is therefore:

```
c.46A>G
```

In this case, the genomic position is translated into a spliced transcript coordinate by accounting for exon boundaries and cumulative coding lengths, as illustrated in Figure 3.2.

Example 2: Intronic variant - downstream offset (c.N+M)

Considering a genomic SNV at position `g.160C>T`, position 160 lies within the intron between Exon 1 and Exon 2 (150–199). The closest coding nucleotide is position 149 (the last coding base of Exon 1). The distance between 149 and 160 is 11 nucleotides. Because the variant lies *downstream* of the last coding base of Exon 1 (moving in the 5'→3' direction of the transcript), it is represented using a positive intronic offset:

```
c.40+11C>T
```

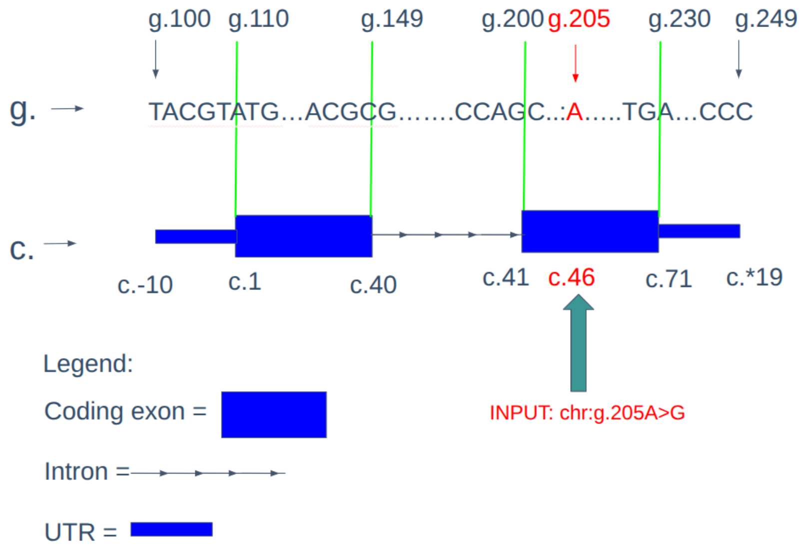

Figure 3.2: **First illustrative example of genomic-to-cDNA mapping.** A genomic variant at position `g.205A>G` is projected onto the spliced transcript coordinate system. The first coding exon contributes positions `c.1`–`c.40`, while the second coding exon continues from `c.41`. The resulting cDNA coordinate is `c.46A>G`.

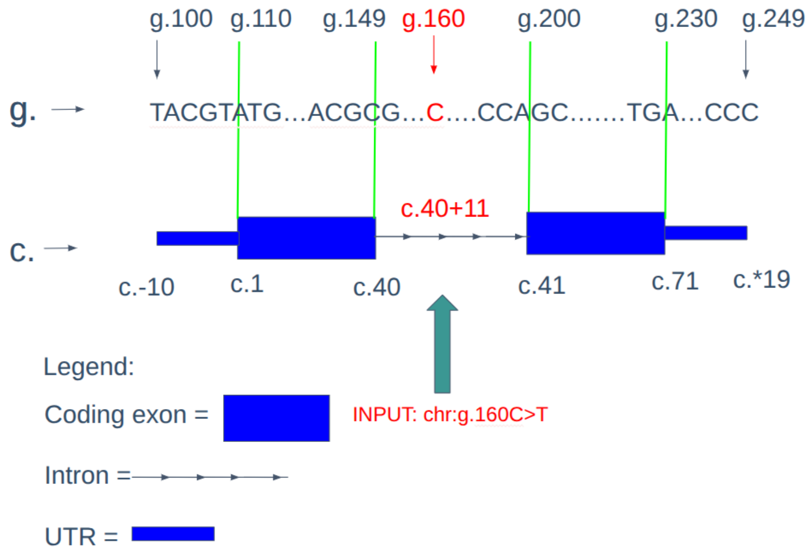

Figure 3.3: **Second illustrative example of genomic-to-cDNA mapping.** A genomic variant at position `g.160C>T` falls within the intronic region between the two exons. The last coding nucleotide of Exon 1 corresponds to `c.40`, and the variant is located 11 nucleotides downstream within the intron. As a result, the cDNA coordinate is expressed as `c.40+11C>T`, following HGVS intronic notation.

Here, `40` corresponds to the last coding nucleotide of Exon 1, and `+11` indicates that the variant lies 11 bases into the downstream intron, as illustrated in Figure 3.3.

Example 3: Intronic variant - upstream offset (c.N−M)

Given a different intronic SNV at position `g.195A>G`, this position also lies within the same intron (150–199), but it is much closer to the beginning of Exon 2 at position 200. Distances from exon boundaries are:

- from 149 (end of Exon 1): 46 nucleotides;
- from 200 (start of Exon 2): 5 nucleotides.

Because the variant is closer to the next exon (Exon 2), its position is expressed relative to the first coding nucleotide of that exon. The first coding base of Exon 2 corresponds to cDNA position 41 (40 coding bases from Exon 1 followed by the first coding base of Exon 2). Since the variant lies 5 bases *upstream* of that exon boundary, it is represented using a negative intronic offset:

```
c.41-5A>G
```

Here, `41` identifies the first coding nucleotide of Exon 2, and `-5` indicates that the variant lies 5 bases upstream within the intron, as illustrated in Figure 3.4.

From cDNA coordinates to protein consequences

When the mapped transcript position lies within a coding sequence (CDS), the workflow proceeds to evaluate potential protein-level consequences. The cDNA position is first interpreted within the CDS coordinate system, where nucleotide positions are grouped into codons (triplets of nucleotides) starting from the translation initiation site (`c.1`). Each codon corresponds to a specific amino acid according to the standard genetic code.

Operationally, the procedure follows three main steps:

1. the cDNA position is converted into a codon index and an intra-codon position. The codon index is computed as $\lceil \text{cDNA position}/3 \rceil$, i.e., by rounding up the division between the nucleotide position and 3 to the nearest integer;

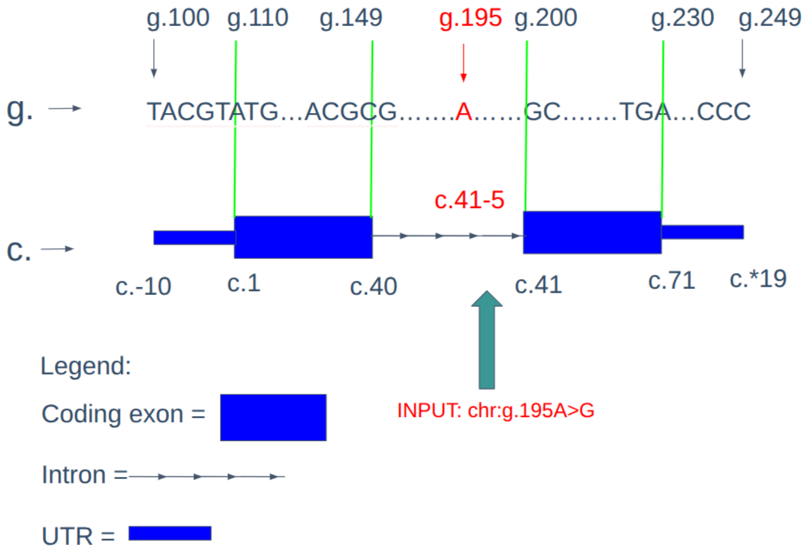

Figure 3.4: **Third illustrative example of genomic-to-cDNA mapping.** A genomic variant at position `g.195A>G` falls within the intronic region between the two exons. The first coding nucleotide of Exon 2 corresponds to `c.41`, and the variant is located 5 nucleotides upstream of this exon boundary. As a result, the cDNA coordinate is expressed as `c.41-5A>G`

2. the reference codon is reconstructed from the CDS sequence;
3. the altered codon (or codon block, in case of indels) is generated by applying the variant.

The reference and altered codons are then translated into amino acids and compared to determine the resulting protein-level effect.

By comparing the reference and altered codons, the system determines whether the variant results in:

- a synonymous substitution (no amino acid change);
- a missense substitution (single amino acid change);
- a multi–amino acid substitution;
- an in-frame insertion or deletion;
- a frameshift leading to downstream sequence alteration;

- a premature stop codon.

Illustrative examples of cDNA-to-protein mapping

Example 1: Missense variant

Considering a coding sequence beginning with:

`ATG GAA TTT CCC`

which is translated as:

Met (p.1) Glu (p.2) Phe (p.3) Pro (p.4)

Supposing a variant occurs at position `c.4G>A`. To interpret this variant:

- `c.4` corresponds to the first nucleotide of the second codon;
- codon index = $\lceil 4/3 \rceil = 2$, i.e., by dividing the nucleotide position by 3 and rounding up to the nearest integer (positions 1–3 map to codon 1, 4–6 to codon 2, etc.);
- intra-codon position = 1 (first base of the codon).

The reference codon is therefore:

`GAA` → Glu

After applying the variant, the codon becomes:

`AAA` → Lys

The resulting protein-level change is:

`p.Glu2Lys`

Example 2: Frameshift variant

Considering the same sequence and a deletion at position `c.4delG`.

The original sequence:

`ATG GAA TTT CCC`

becomes:

`ATG AAT TTC CC...`

To interpret the effect:

- `c.4` lies in the second codon;
- deletion shifts the reading frame starting from codon 2;
- all downstream codons are redefined.

The new codon structure becomes:

`ATG AAT TTC ...`

which translates into a completely different amino acid sequence from position 2 onward, typically leading to a premature stop codon downstream.

This is represented using HGVS frameshift notation:

`p.Glu2AsnfsTerX`

where `fs` indicates a frameshift and `TerX` denotes the position of the newly introduced stop codon.

Integration of external variant resources

In addition to transcript-derived validations, the dbSNP variant files can be interrogated (if present among the loaded resources). When a match is found, dbSNP identifiers are appended to the validation record. As already mentioned in Par. 3.2.1, these external resources do not influence coordinate mapping or consequence inference; they function exclusively as supplementary validation layers that enrich the final output.

Final output structure

The final output is constructed as a collection of validation records derived from the same input variant. Each record corresponds to a specific transcript or genomic context and includes:

- the normalized genomic representation of the variant;
- the associated gene and transcript identifiers;
- region classification (e.g., exonic, intronic);
- transcript-level and, when applicable, protein-level consequences;
- optional dbSNP identifier.

The overall genomic input workflow is shown in Figure 3.5.

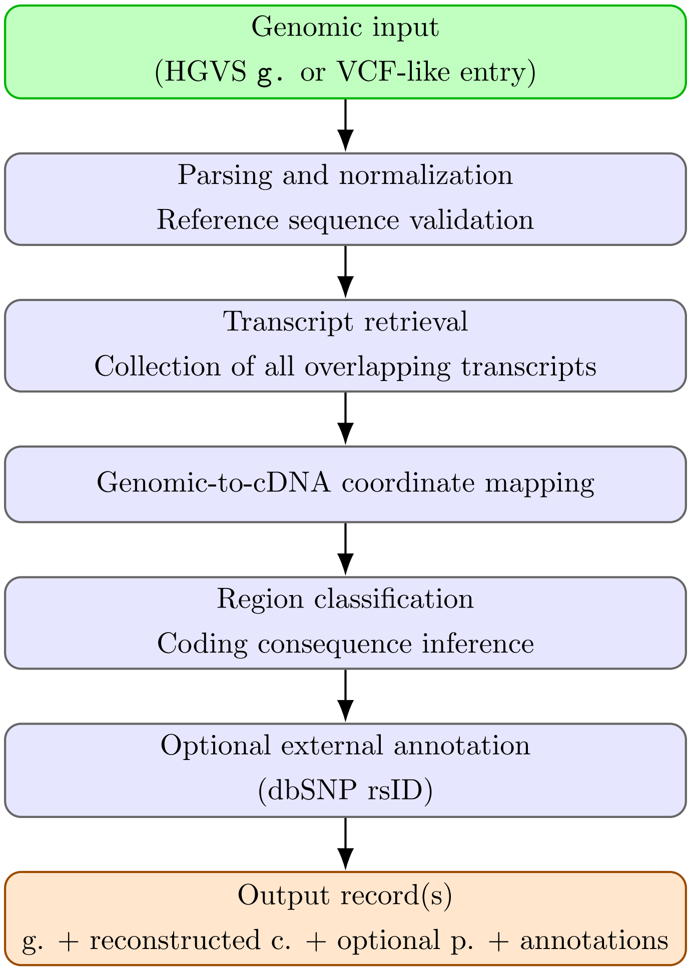

Figure 3.5: **Genomic input validation workflow.** A genomic variant provided in HGVS `g.` notation or VCF format undergoes parsing and normalization, including chromosome harmonization and reference allele validation against the indexed genome assembly. The normalized variant is then mapped onto all overlapping transcripts retrieved from the internal pre-processed database. For each compatible transcript, genomic coordinates are projected to cDNA and, when applicable, protein levels, enabling region classification and functional consequence inference. Optional external resources such as dbSNP are queried using the reconstructed genomic coordinates, and matching identifiers are appended to the output. Because multiple transcripts may overlap the same genomic locus, a single input variant can generate multiple transcript-specific validation records.

3.2.2.2 Coding DNA input workflow (c.)

Coding DNA variants (`c.`) describe nucleotide alterations relative to the coding sequence of a specific transcript, where positions are numbered starting from the translation initiation codon (`c.1`) and follow the spliced transcript structure. Variants expressed at this level are processed through the TransVar `canno` command-line, which activates the cDNA-driven validation workflow.

Unlike genomic input, where chromosome coordinates represent the primary reference frame, coding DNA input is interpreted relative to transcript models stored in the internal TransVar database.

Input parsing and transcript resolution

A cDNA variant is first parsed into an internal structure containing (i) the transcript or gene identifier, (ii) the HGVS coding position within the spliced transcript, and (iii) the reference and alternate nucleotide sequence describing the alteration.

Two distinct input scenarios are supported:

- transcript-level input (e.g., `NM_001407571.1:c.123A>G`), where the transcript identifier is explicitly provided;
- gene-level input (e.g., `BRCA1:c.123A>G`), where only the gene name is specified.

Transcript resolution is performed through the name-based indexing mechanism described in Par. 3.2.1, enabling direct retrieval of transcript records using either transcript identifiers or gene symbols as input.

In the transcript-level input case, the workflow resolves the exact transcript within the selected annotation database and proceeds with a single transcript context.

In the gene-level input case, all transcripts associated with the specified gene in the loaded database are examined. However, only those transcripts for which the provided cDNA coordinate is valid and compatible with the transcript structure are retained. For example, transcripts are excluded from the final output if the specified position cannot be mapped due to differences in exon organization, or if the reference nucleotide at the resolved position does not match the reference allele provided in the input.

Consequently, transcript-specific input typically produces a single validation record, whereas gene-level input may generate multiple records (one per each compatible transcript) reflecting alternative exon structures, coding boundaries, and transcript versions. Transcripts for which the provided cDNA coordinate is not biologically consistent are not reported.

Interpretation of cDNA coordinates

Once the relevant transcript(s) are identified, the cDNA coordinate is interpreted relative to the exon structure of each transcript. Positions located within exons correspond directly to coding DNA coordinates, while positions expressed with intronic offsets are resolved relative to exon-intron junctions according to HGVS conventions. Because transcript models are stored as ordered exon segments with defined coding start and end positions, the system can determine whether the variant lies within the coding sequence (CDS) or other spaces.

Projection from cDNA to genomic coordinates

Once the cDNA position has been determined within the transcript, it is mapped back onto the genomic coordinates using the stored exon boundaries. This process mirrors the logic used in the genomic-to-transcript conversion (`ganno`), but in a reverse way, projecting transcript positions onto their corresponding genomic loci. Strand orientation is explicitly considered during this projection, as it was for the genomic workflow. As in the genomic workflow, multi-nucleotide substitutions and complex events undergo normalization procedures.

Illustrative examples of cDNA-to-genomic mapping

To clarify how cDNA coordinates are converted into genomic coordinates, the same simplified transcript model defined in 3.2.2.1 located on the positive strand is considered:

- the first exon, including a 5' UTR segment and a coding exon, has genomic location 100–149;

- the second exon, including a coding exon and a 3' UTR segment, has genomic location 200–249;
- the coding sequence (CDS), beginning with the Start Codon ATG, starts in the genomic position 110 and includes all the coding exons of the gene (in this example, up to the genomic position 230).

Example 1: Exonic variant

Supposing a cDNA SNV is provided as:

`c.46A>G`

the system must determine the corresponding genomic position.

To do so, the cDNA coordinate is first interpreted within the CDS: exon 1 (110–149) contributes 40 coding bases, therefore `c.1`–`c.40` map to exon 1. The remaining positions are located in exon 2.

The queried position is:

$$46 - 40 = 6$$

Thus, `c.46` corresponds to the 6th coding nucleotide within exon 2.

Because exon 2 starts at genomic position 200, the genomic coordinate is:

$$200 + 6 - 1 = 205$$

The resulting genomic representation is therefore:

`g.205A>G`

Example 2: Intronic variant - downstream offset (c.N+M)

Considering a cDNA-level variant:

`c.40+11C>T`

Here, `c.40` corresponds to the last coding nucleotide of exon 1 (position 149), and `+11` indicates that the variant lies 11 nucleotides downstream within the intron. The genomic coordinate is therefore computed as:

$$149 + 11 = 160$$

The corresponding genomic representation is:

`g.160C>T`

Example 3: Intronic variant - upstream offset (c.N−M)

Given a cDNA-level variant:

`c.41-5A>G`

`c.41` corresponds to the first coding nucleotide of exon 2, which maps to genomic position 200. The `-5` indicates that the variant lies 5 nucleotides upstream of this exon boundary within the intron.

The genomic coordinate is therefore:

$$200 - 5 = 195$$

The resulting genomic representation is:

`g.195A>G`

In all cases, transcript-to-genomic mapping is performed by resolving the position of the cDNA coordinate within the ordered coding exons and projecting it back onto the genome using exon boundaries and strand information.

Once the cDNA position has been projected onto the corresponding genomic locus, the reference nucleotide is validated against the reference genome sequence. In particular, the system retrieves the nucleotide at the resolved genomic position from the indexed FASTA file and interprets it in the strand context of the transcript.

Protein-level consequence inference

The procedure is the same as the one described in the genomic workflow (Par. 3.2.2.1). If the resolved cDNA position falls within the coding sequence, the altered nucleotide is evaluated within its codon context.

Integration of dbSNP resources

As in the genomic workflow, when a cDNA match is identified and the genomic coordinates are reconstructed, dbSNP identifiers are appended to the output record.

Output structure

Each output record derived from a cDNA input includes:

- the original cDNA representation of the variant;
- the resolved transcript and gene identifiers;
- the reconstructed genomic description;
- region classification within the transcript;
- protein-level consequence when applicable;
- optional dbSNP identifiers.

Thus, transcript-specific input produces a single primary validation, while gene-level input may generate multiple transcript-dependent records, reflecting the biological diversity of transcript isoforms associated with the same gene.

The cDNA validation workflow is summarized in Figure 3.6.

3.2.2.3 Protein input workflow (p.)

Protein variants (`p.`) describe amino acid alterations relative to the translated product of a specific transcript, where positions are numbered starting from the first amino acid of the protein sequence. Variants expressed at this level are processed through the TransVar `panno` command-line, which activates the protein-driven validation workflow.

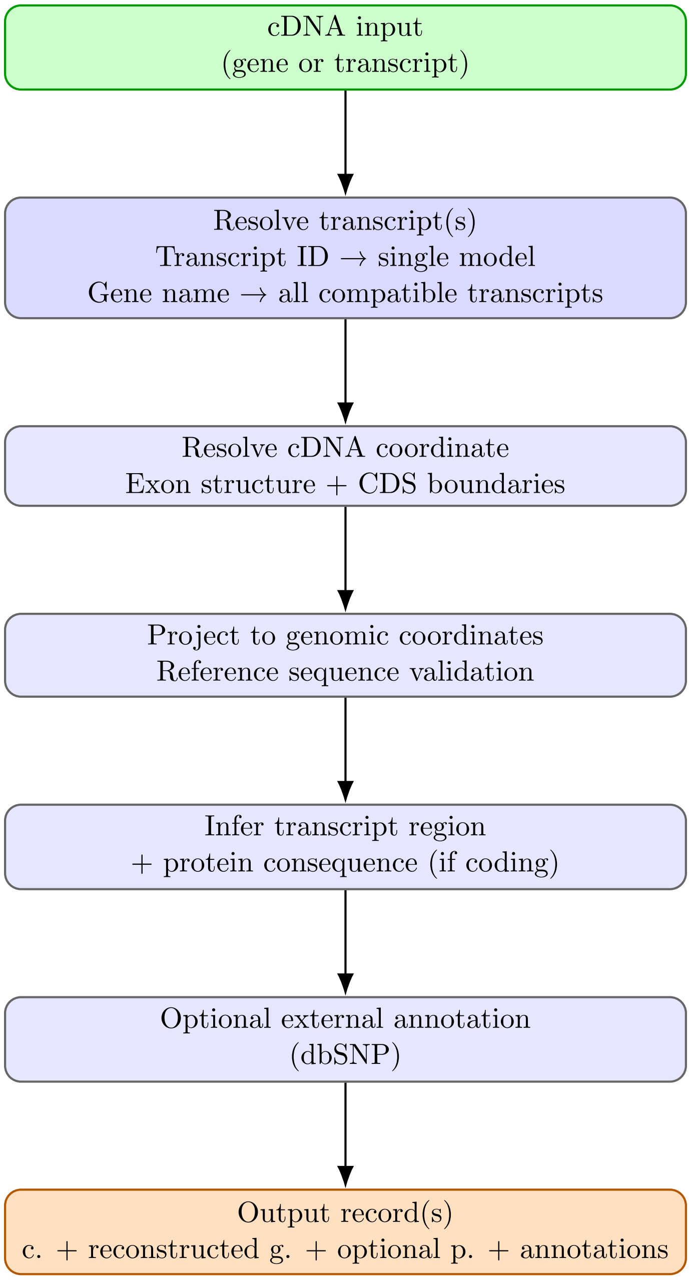

Figure 3.6: **cDNA input validation workflow.** A cDNA variant is parsed and resolved against one or multiple transcript models. The coding coordinate is interpreted relative to exon structure and CDS boundaries, projected to genomic space, functionally classified at transcript and protein level, optionally enriched with external resources such as dbSNP, and finally emitted as one or multiple transcript-dependent output records.

Unlike genomic and cDNA workflows-where nucleotide coordinates provide a direct mapping framework-protein-level input requires an additional inference step, as multiple nucleotide sequences may encode the same amino acid change.

Input parsing and transcript resolution

A protein variant is first parsed into an internal representation capturing the transcript or gene identifier, amino acid position, and altered residue. As in the cDNA workflow, transcript resolution is performed through the name-based indexing mechanism described in Par. 3.2.1, enabling direct retrieval of transcript records using either transcript identifiers or gene symbols. Two input modalities are supported:

- transcript-specific input (e.g., `NM_007294.4:p.His41Gly`), where a precise transcript is defined;
- gene-level input (e.g., `BRCA1:p.His41Gly`), where only the gene name is provided.

In the transcript-specific case, interpretation is restricted to the indicated transcript model. In the gene-level case, all transcripts associated with the specified gene are examined; however, only those transcripts whose coding sequence includes the specified amino acid position are retained. Transcripts having a different amino acid or lacking a compatible coding region are excluded.

Protein-to-cDNA inference

Because the input describes an amino acid change rather than a nucleotide change, the system must infer which codon(s) could produce the observed protein alteration. Each amino acid is encoded by one or more codons, and therefore a single protein-level variant may correspond to multiple possible nucleotide substitutions.

For a given transcript, the amino acid position is first converted into its corresponding codon position within the coding sequence.

The reference codon is reconstructed from the transcript sequence by retrieving the underlying nucleotide sequence and translating it according to the genetic code.

The resulting amino acid is then compared with the reference residue provided in the input protein variant. If the translated amino acid does not match the expected reference residue, the transcript is considered incompatible and is excluded from further processing.

Only for compatible transcripts, alternative codons consistent with the requested amino acid change are then enumerated.

As discussed in Par. 2.3.1, TransVar resolves the ambiguity introduced by codon degeneracy using a parsimony-based strategy. For each candidate codon, the number of nucleotide substitutions required to convert the reference codon into the alternative codon is computed. The candidate requiring the smallest number of nucleotide changes is selected as the primary representation used for subsequent projection to cDNA and genomic coordinates.

At the same time, additional codon combinations also compatible with the requested amino acid substitution are retained and reported as alternative candidate events in the output. This approach makes the inherent ambiguity of protein-level input explicit while avoiding arbitrary selection of a single nucleotide interpretation.

Illustrative example of protein-to-cDNA inference

To clarify how protein-level variants are translated into cDNA coordinates, a simplified transcript is considered whose coding sequence is interpreted starting from `c.1`, corresponding to the first nucleotide of the start codon.

Supposing a protein-level variant is provided as:

`p.Glu10Lys`

Each amino acid is encoded by a codon consisting of three nucleotides. Therefore, amino acid position n corresponds to the following cDNA interval:

$$\texttt{c.}(3n-2)\texttt{--c.}(3n)$$

For position $n = 10$:

$$\texttt{c.}(3 \cdot 10 - 2) = \texttt{c.28}$$

$$\texttt{c.}(3 \cdot 10) = \texttt{c.30}$$

Thus, amino acid position 10 corresponds to the codon spanning:

$$\texttt{c.28--c.30}$$

The nucleotide triplet at positions `c.28--c.30` is extracted from the CDS. Supposing the reference codon is:

$$\texttt{GAA} \rightarrow \text{Glu (p.10)}$$

Lysine (Lys) can be encoded by:

$$\texttt{AAA}, \quad \texttt{AAG}$$

Thus, multiple nucleotide changes can produce the same amino acid substitution.

The number of nucleotide substitutions required for each candidate is evaluated:

- `GAA` → `AAA`: 1 substitution (at `c.28`)
- `GAA` → `AAG`: 2 substitutions (at `c.28` and `c.30`)

The minimal-change candidate is selected:

$$\texttt{c.28G>A}$$

Projection to genomic coordinates

Once candidate codon changes are determined and are translated into cDNA coordinates, genomic coordinates are reconstructed using the exon structure and strand orientation, as described for the coding DNA input workflow (Par. 3.2.2.2).

Strand awareness remains essential: for transcripts on the negative strand, nucleotide substitutions are reverse-complemented to ensure consistency with the reference genome assembly.

Functional interpretation and output

Because the input already specifies a protein alteration, the primary functional consequence is intrinsically defined. However, the workflow validates that the inferred nucleotide change is consistent with the transcript coding sequence and determines whether additional effects, such as multi–amino acid substitutions or frameshift implications, are implied.

Each output record includes:

- the original protein-level representation;
- the resolved transcript and gene identifiers;
- the inferred cDNA description;
- the reconstructed genomic coordinates;
- confirmation of coding consequence consistency.

The protein validation workflow is shown in Figure 3.7.

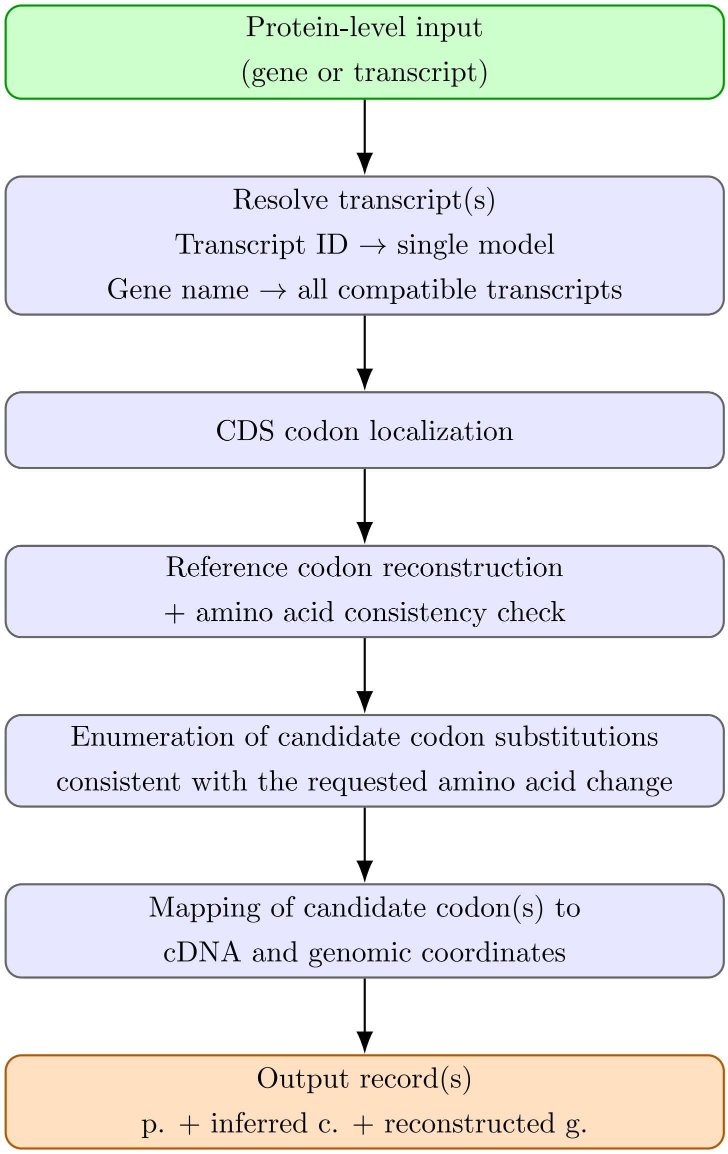

Figure 3.7: **Protein input validation workflow.** A protein-level variant is first resolved to one or more compatible transcript models. The corresponding codon position within the CDS is identified, and the reference codon is reconstructed to verify consistency with the amino acid specified in the input. For compatible transcripts, candidate nucleotide substitutions capable of producing the requested amino acid change are then enumerated. Each candidate codon alteration is mapped onto spliced cDNA coordinates and projected to genomic coordinates using exon structure and strand orientation. The resulting records integrate protein, transcript, and genomic representations into a unified validation output.

3.2.3 Limitations and methodological improvements

Following the systematic validation of the original TransVar implementation, several limitations were identified, differing in scope and impact on validation accuracy and consistency. Some issues involved localized inconsistencies and lacknesses, whereas others revealed more structural constraints affecting coordinate interpretation and variant classification logic.

The methodological improvements introduced in this work are therefore organized into two categories: minor refinements, addressing specific technical inconsistencies or lacknesses without altering the core architecture, and major improvements, involving modifications to coordinate handling.

Minor improvements

A first limitation identified in the original TransVar preprocessing strategy concerned the selection of transcript types retained within the internal `.transvardb` datasets. By design, the original implementation stored primarily protein-coding transcripts. Non-coding transcripts (NR/ENST), including long non-coding RNAs (lncRNAs), non-coding RNAs (ncRNAs) and antisense transcripts were systematically excluded during GTF and GFF parsing and database serialization.

This choice introduced two practical limitations. First, several genes exclusively associated with non-coding transcripts (e.g., *NEAT1*) were entirely absent from the internal validation database. Second, gene-level queries did not reflect the full transcript diversity provided by the reference validation source.

To address these limitations, the preprocessing logic was modified to retain all transcript categories associated with each gene, including both coding and non-coding accessions. As a result, the internal `.transvardb` files now preserve a complete transcript representation consistent with the original GFF/GTF sources.

The inclusion of non-coding transcripts required not only their retention during preprocessing, but also a consistent strategy for coordinate mapping within an engine originally designed around coding DNA (c.) nomenclature.

According to HGVS recommendations, non-coding transcripts must be described using the `n.` prefix rather than `c.`. In this nomenclature standard, nucleotide positions are counted from the first transcribed nucleotide of the RNA molecule, not from a translation initiation codon. Unlike coding transcripts, there is no reference to a CDS start. Position `n.1` corresponds to the first nucleotide of the spliced transcript sequence.

An example of a non-coding transcript serialized in the internal `.transvardb` format is shown in Table 3.2. This illustrates how non-coding transcripts are represented within the same transcript-centric structure adopted for protein-coding transcripts, while preserving their exon–intron organization.

Field	**Value**
Gene symbol	FGFR2
Transcript ID	NR_073009
Transcript version	2
Transcript biotype	ncRNA
Transcript span	123237846–123357958
Chromosome	chr10
Strand	-
CDS boundaries	123237846–123357958
Exon coordinates	[(123237846,123239535), ... , (123357476,123357958)]
Protein accession	—
External identifiers	GeneID:2263, HGNC:3689, MIM:176943

Table 3.2: Example of transcript-centric record stored in the internal `.transvardb` database for the gene *FGFR2*.

Because non-coding transcripts do not contain an annotated CDS interval, a conceptual difficulty arises: the internal coordinate system of TransVar is intrinsically CDS-centric, meaning that transcript coordinates are normally anchored to CDS boundaries.

To avoid introducing a separate mapping engine for non-coding RNAs, the entire transcript span is internally treated as a single "coding-like" interval. In practice, the CDS boundaries are set equal to the transcript start and end, while the original exon structure is preserved unchanged. As a result, the internal coordinate system spans the full spliced transcript sequence, while still reflecting the true exon–intron organization of the transcript.

Conceptually, this representation is not biologically accurate, since no translation occurs. However, from a coordinate perspective it is equivalent to the HGVS `n.` model, where nucleotide positions are counted from the beginning of the spliced transcript across exons in transcriptional order.

In this way, the numerical indexing remains identical to what HGVS prescribes for `n.` notation.

This design choice ensures that genomic-to-transcript (g→n) mapping remains exon-aware and strand-aware, and that transcript-to-genomic (n→g) projection uses the same robust exon-based logic implemented for coding transcripts.

The only inconsistency left lies in the HGVS prefix. Since the internal engine historically produces `c.` coordinates, a post-processing adjustment was introduced. After mapping, the transcript biotype is inspected:

- if the transcript is non-coding, the `c.` prefix must be replaced with `n.`;
- if the transcript is protein-coding, the standard `c.` notation must be retained.

Finally, an additional enhancement was introduced to incorporate transcript prioritization metadata directly into the `.transvardb` structure.

The MANE (Matched Annotation from NCBI and Ensembl) project was established to define a single, representative transcript per protein-coding gene that is consistently annotated between RefSeq and Ensembl. A *MANE Select* transcript represents the primary, high-confidence reference transcript agreed upon by both annotation consortia, with identical exon structure, coding sequence, and genomic coordinates across databases. In addition, *MANE Plus Clinical* transcripts extend this framework by including alternative transcripts that are clinically relevant, particularly when known pathogenic variants occur outside the MANE Select isoform [54].

For GRCh38 (both RefSeq and Ensembl), MANE Select and MANE Plus Clinical annotations were integrated into the preprocessing stage using the MANE-related attributes which were already present within the source GTF/GFF annotation files. For GRCh37 RefSeq datasets, analogous RefSeq Select and Plus Clinical information flags were incorporated to ensure consistent transcript prioritization across genome builds.

These annotations are now stored at preprocessing time and propagated to runtime outputs. This enables deterministic transcript prioritization according to clinically relevant criteria. Specifically:

- for GRCh38 (both RefSeq and Ensembl): MANE Select → MANE Plus Clinical → other protein-coding transcripts → non-coding transcripts;
- for GRCh37 (RefSeq): RefSeq Select → RefSeq Plus Clinical → other protein-coding transcripts → non-coding transcripts;
- for GRCh37 (Ensembl): Protein-coding transcripts → non-coding transcripts.

This prioritization logic is consistently applied across g., c., and p. workflows. As a consequence, validation outputs are now both more informative and more clinically aligned.

Major improvements

A major limitation identified during validation concerns the mapping of variants located within 5' and 3' untranslated regions (UTRs). A comparative analysis against SnpEff revealed systematic discrepancies affecting both coordinate representation and regional interpretation for this variant category.

Variants located in untranslated regions can belong to two distinct categories. *Exonic UTR variants* fall within UTR segments of exons (non-coding portions of exons located in the 5' or 3' untranslated regions) and are expressed using HGVS coordinates such as `c.-N` or `c.*N` with N indicating the exonic offset.

In contrast, *intronic UTR variants* occur within introns adjacent to UTR exons and are represented using compound expressions such as `c.-N±M` or `c.*N±M`, where the intronic offset M is measured relative to the nearest UTR exon boundary.

When identical genomic-level inputs are provided, TransVar and SnpEff produce divergent transcript-level outputs. Representative examples include:

- Input: `chr10:g.100177309G>A` ⇒ TransVar: `c.2103+12C>T` vs SnpEff: `c.*12C>T` for transcript `NM_000195`

- Input: `chr19:g.44905796C>T` ⇒ TransVar: `c.1-829C>T` vs SnpEff: `c.-69C>T` for transcript `NM_000041`
- Input: `chr11:g.66290982delA` ⇒ TransVar: `c.786+3445delA` vs SnpEff: `c.*22-2046delA` for transcript `NM_001348571`

Although the genomic coordinates are identical, the resulting `c.` descriptions differ substantially. SnpEff reports canonical HGVS UTR notation (`c.-N` for 5'UTR and `c.*N` for 3'UTR), whereas the original TransVar implementation expresses these positions as linear offsets relative to CDS boundaries (e.g., `c.1-N` or `c.<CDSlen>+N`).

This discrepancy originates from a structural design choice in the original TransVar architecture. During preprocessing, transcript sequences are reconstructed in a CDS-centric manner: the internal spliced coordinate system is built exclusively on the concatenated coding exons (CDS), rather than on the full spliced transcript including 5'UTR and 3'UTR exonic segments. Exon–intron boundaries are correctly stored and therefore canonical exonic and intronic variants within the CDS (e.g., `c.25+3`) are handled appropriately. However, untranslated regions are not incorporated into the indexed transcript coordinates array used for cDNA mapping.

As a consequence, the internal coordinate system is intrinsically anchored to CDS boundaries. When a variant falls outside the CDS, TransVar does not compute its position along the full spliced transcript. Instead, it measures a linear genomic offset from either the CDS start or the CDS end and encodes this distance relative to `c.1` (for upstream positions) or to the last CDS nucleotide (for downstream positions). This produces representations such as `c.1-19053` or `c.2103+12`, which resemble intronic-style offsets rather than canonical UTR notation.

In contrast, HGVS-compliant transcript-aware annotators reconstruct the entire spliced transcript (5'UTR + CDS + 3'UTR) and calculate UTR coordinates using transcript-based distances. Thus, `c.-346` reflects 346 nucleotides upstream of the CDS start counted along concatenated 5'UTR exons, while `c.*12` reflects 12 nucleotides downstream of the stop codon along concatenated 3'UTR exons.

Importantly, this limitation affects both genomic-to-transcript projection ($g \rightarrow c$) and transcript-to-genomic mapping ($c \rightarrow g$). The issue is therefore bidirectional and rooted in the coordinate model itself.

Thus the absence of a UTR-aware spliced coordinate system prevents the generation of canonical HGVS UTR expressions. Consequently, variants located within 5' and 3' UTR regions are systematically represented using CDS-relative genomic offsets, leading to frequent discrepancies during cross-tool validation. A clarifying example of the different validation workflow followed by TransVar and by SnpEff for this category of variants is shown below in details.

Considering the genomic variant:

`chr19:g.44905796C>T`

for transcript `NM_000041` (strand = `+`), whose CDS spans `[44906625,44909250]` and whose annotated 5'UTR interval is `[(44905796,44905841), (44906602,44906624)]`.

The variant position `g.44905796` lies within the 5'UTR region.

In the original TransVar implementation, transcript coordinates are indexed only along the spliced CDS.

Thus, the first CDS nucleotide is internally represented as `c.1`. For positions upstream of the CDS, TransVar computes a genomic offset from the CDS start:

$$N = \texttt{cds\_beg - gpos} = 44906625 - 44905796 = 829$$

Rather than using HGVS upstream notation, the position is expressed as an offset from the first CDS nucleotide:

`c.1-829`

The final output becomes:

`NM_000041:c.1-829C>T`

HGVS-compliant validators instead reconstruct the full spliced transcript, including UTR segments, and compute coordinates relative to the transcript structure. For a positive-strand transcript, the last nucleotide immediately upstream of the CDS corresponds to:

$$\texttt{c.-1} \iff \texttt{g.(CDS\_beg - 1)} = \texttt{g.44906624}$$

The upstream index N is then determined by counting nucleotides along the spliced 5'UTR sequence. Because the 5'UTR is distributed across multiple exonic segments, the transcript-level distance does not coincide with the genomic distance.

By traversing the ordered 5'UTR exon blocks, the variant position corresponds to:

$$N = 69$$

This yields the canonical HGVS 5'UTR notation:

`NM_000041:c.-69C>T`

Both TransVar and SnpEff correctly identify the variant as upstream of the CDS. However, TransVar anchors the coordinate to the first CDS nucleotide and reports a CDS-relative offset (`c.1-829`), whereas SnpEff reports a transcript-spliced 5'UTR position using canonical HGVS notation (`c.-69`).

This outcome is systematically produced whenever the variant falls outside the CDS in the original TransVar logic: rather than numbering UTR positions using HGVS notation, the tool anchors coordinates to the terminal CDS index (`CDSlen`) and expresses downstream positions (such as `CDSlen+N`) or upstream positions (such as `1-N`).

This difference is clearly illustrated in Figure 3.8, where the discrepancy between genomic offset and transcript-spliced distance shown in the previous example becomes evident.

Implementation of UTR-aware validation for exonic and intronic variants

Since the original TransVar implementation was not UTR-aware, untranslated regions were not explicitly modeled within the internal transcript coordinate system. All sequence outside the CDS was effectively treated as generic non-coding space.

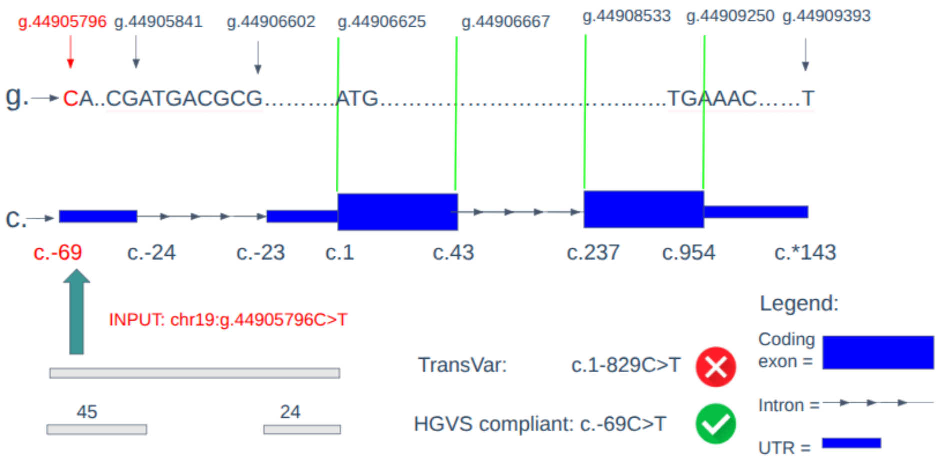

Figure 3.8: **UTR mapping discrepancy between TransVar and HGVS-compliant annotation.** Graphical representation of the variant `chr19:g.44905796C>T` mapped onto the transcript `NM_000041`. In the original TransVar implementation, transcript coordinates are computed relative to the CDS only, resulting in a genomic offset from the CDS start (`c.1-829`). In contrast, a HGVS-compliant annotator computes the position along concatenated 5'UTR exonic segments, producing `c.-69`.

To solve this limitation, UTR segments were explicitly reconstructed during preprocessing and stored for each transcript within the `.transvardb` internal database.

For every protein-coding transcript, 5' and 3' untranslated regions were explicitly reconstructed during preprocessing using the annotated exon coordinates together with the CDS boundaries. Rather than treating UTRs as single contiguous genomic intervals, the reconstruction follows the transcript exon structure.

Protein-coding transcripts are composed of multiple exons separated by introns. Because the CDS may begin or end within an exon, untranslated regions often correspond to non-coding segments of exons and may be distributed across several non-contiguous segments. Consequently, UTRs must be represented as a collection of exon-derived segments rather than as a single continuous region.

To address this, the preprocessing step iterates over the exon list of each transcript and compares every exon with the CDS boundaries. For each exon, three situations may occur:

- the exon lies entirely upstream of the CDS start and therefore contributes completely to the 5'UTR;
- the exon lies entirely downstream of the CDS end and therefore contributes completely to the 3'UTR;
- the exon overlaps the CDS boundary, in which case only the non-coding portion of the exon is retained as UTR.

The resulting segments are stored as ordered lists of genomic intervals. All exonic portions located before the CDS start are serialized as `utr5` blocks, while segments located after the CDS end are stored as `utr3` blocks. Because these segments originate from exons, they represent the spliced transcript structure and automatically preserve strand orientation and exon ordering.

These segmented regions are referred to as *UTR blocks*.

Considering the protein-coding transcript `NM_001351992` shown below:

```
NM_001351992   chr11   strand = +
Transcript span:  93474838 -- 93497915
CDS: 93480560 -- 93494854
Exons:  [(93474838,93474894), (93480468,93480614), (93483512,93483610),
         (93486848,93486921), (93487102,93487203), (93490482,93490631),
         (93492908,93493024), (93494681,93497915)]
```

The CDS begins inside the second exon. Consequently:

- the first exon `(93474838,93474894)` lies entirely upstream of the CDS and therefore belongs completely to the 5'UTR;
- the second exon `(93480468,93480614)` overlaps the CDS start, so only the portion preceding the CDS boundary contributes to the 5'UTR;
- the following exons are entirely coding until the last exon;
- the last exon `(93494681,93497915)` overlaps the CDS end, so only the segment located after the CDS boundary contributes to the 3'UTR.

After processing all exons, the resulting UTR segments are stored internally as ordered genomic blocks:

```
utr5 = [(93474838,93474894), (93480468,93480559)]
utr3 = [(93494855,93497915)]
```

By serializing UTRs as exon-derived blocks, the transcript model becomes fully structure-aware. Distances within UTR regions can now be computed along the spliced transcript, enabling correct handling of both exonic and intronic UTR variants according to HGVS coordinate conventions.

Revised genomic-to-cDNA mapping (g → c)

When a genomic coordinate is provided, the mapping procedure now first checks whether the position falls within a stored UTR block. If the coordinate lies inside an exonic UTR block, the transcript offset is computed by cumulatively traversing the ordered UTR exon blocks starting from the CDS-adjacent UTR exon and proceeding in transcript direction, counting only nucleotides belonging to UTR exons and excluding intronic sequence. This produces canonical HGVS expressions such as `c.-N` or `c.*N`, where N represents the distance measured along the spliced UTR transcript sequence.

Otherwise, if the coordinate lies in an intron between two UTR exons, the nearest UTR exon boundary is identified. The transcript-level offset N is first determined for that exon boundary by traversing the ordered UTR exon blocks, and the intronic displacement M is then computed relative to the exon edge, producing `c.-N±M` or `c.*N±M`.

Revised cDNA-to-genomic mapping (c → g)

The same UTR-aware logic was implemented in the reverse direction in order to support canonical projection of transcript-level UTR coordinates back to genomic space.

When the input coordinate corresponds to an exonic UTR position (e.g., `c.-N` or `c.*N`), the value of N is interpreted as a transcript-relative distance measured along the ordered UTR exon blocks. The mapping procedure therefore traverses the corresponding `utr5` or `utr3` blocks in transcriptional order, cumulatively subtracting exon lengths until the block containing the requested offset is identified. Once the correct UTR exon block has been located, the exact genomic nucleotide is recovered within that block in a strand-aware manner.

When the input coordinate corresponds to an intronic UTR position (e.g., `c.-N±M` or `c.*N±M`), the mapping is performed in two consecutive steps. First, the exonic anchor associated with N is resolved exactly as for an exonic UTR coordinate, thus identifying the UTR exon boundary to which the HGVS notation refers. Second, the intronic displacement M is applied relative to that exon edge, following transcript orientation and strand. In this way, the genomic position is not derived from a direct CDS-relative offset, but from a combination of (i) transcript-spliced localization of the UTR anchor and (ii) local intronic displacement from the appropriate exon boundary.

Importantly, this revised logic also extends reference-allele validation to UTR coordinates. Once the genomic position has been reconstructed, the reference nucleotide specified in the input is checked against the corresponding base in the reference genome assembly. Thus, the method does not only recover the correct genomic locus for a UTR coordinate, but also validates whether the submitted HGVS expression is consistent with the underlying assembly sequence.

Through all these modifications, the transcript model evolved from a CDS-relative coordinate system into a structure-aware representation that explicitly incorporates segmented UTR exons. UTR variants are now correctly distinguished from intronic regions, and their numbering follows HGVS rules by counting along the spliced transcript rather than across genomic intervals.

As a result, both genomic-to-transcript and transcript-to-genomic projections are now consistent, strand-aware, and fully compliant with canonical UTR notation for both exonic and intronic variants.

The new UTR-aware bidirectional variant validation is summarized in Figure 3.9.

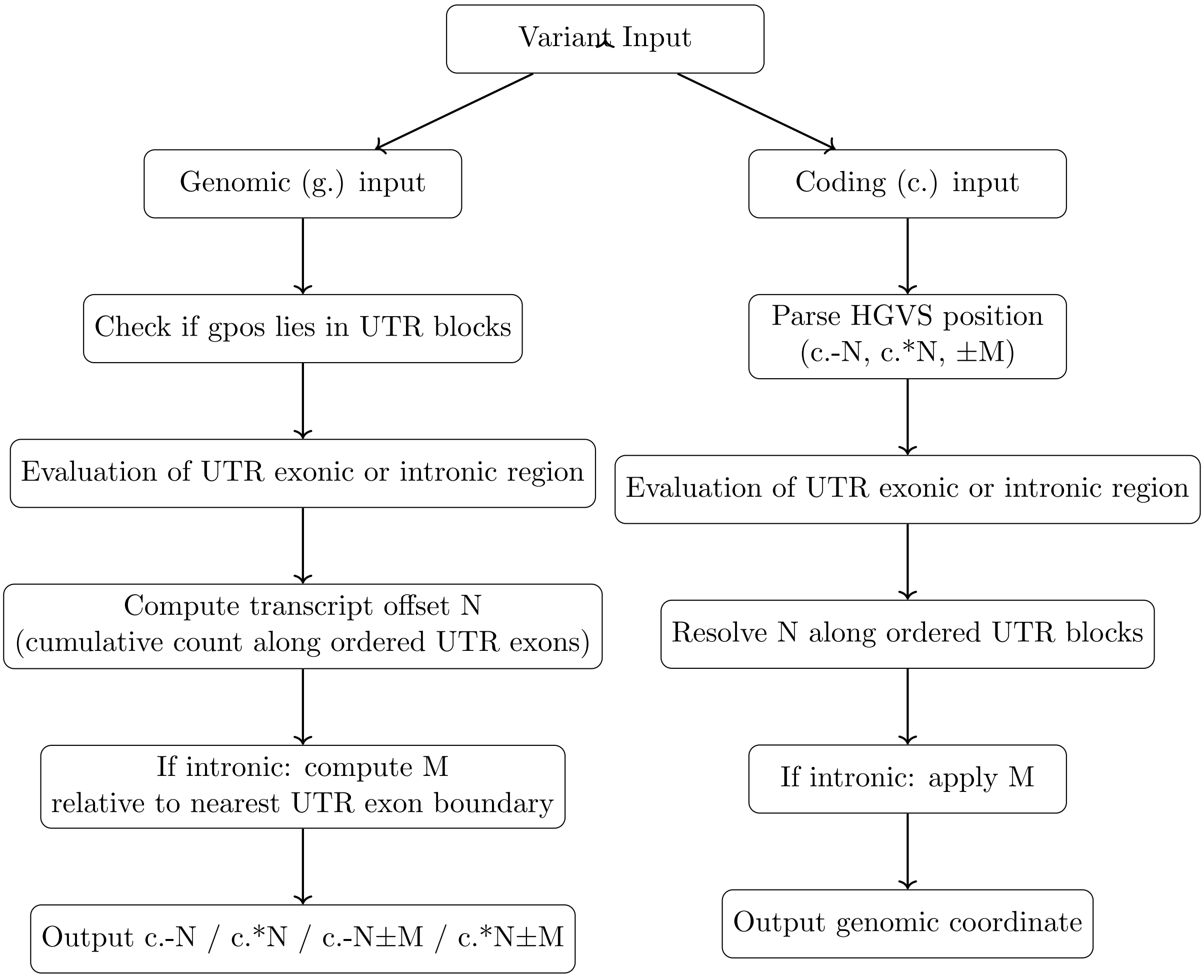

Figure 3.9: **UTR-aware bidirectional mapping workflow.** The revised implementation relies on explicit storage of segmented UTR exon blocks. Both genomic-to-transcript (g→c) and transcript-to-genomic (c→g) projections distinguish exonic and intronic UTR variants. Transcript offsets are computed cumulatively along ordered UTR exons, while intronic displacements are applied relative to exon boundaries in a strand-aware manner.

3.2.4 rsID validation integration

A relevant limitation of the original TransVar implementation concerns the accepted input formats. In particular, TransVar does not support dbSNP identifiers (rsID) as direct input. The tool can only processes genomic variants expressed either in HGVS genomic notation (g.) or in VCF-like format, through the `ganno` command whose workflow has been described in Par. 3.2.2.1.

However, in clinical and research practice, variants are frequently referenced by their dbSNP identifiers (e.g., rsIDs), which represent stable and widely used accession numbers in public databases.

To overcome this issue, an initial strategy was designed to dynamically retrieve the corresponding genomic coordinates starting from a provided rsID. The underlying idea was to query the official dbSNP VCF files (hg19 and hg38 builds), saved in the internal TransVar database, and extract the matching VCF genomic record, which could then be reformatted and passed as input to the `ganno` workflow. This extraction step was implemented using standard command-line tools such as `grep` and `bcftools`, which allow filtering and querying large VCF files.

Although conceptually straightforward, this approach proved to be computationally inefficient. The dbSNP VCF files for GRCh37 and GRCh38 are extremely large, containing hundreds of millions of variants and occupying several gigabytes of disk space. Even when indexed and queried with optimized tools, each rsID lookup requires scanning large portions of the file. As a consequence, the time elapsed between rsID submission and retrieval of the corresponding genomic variant may reach several minutes. This latency significantly affects the usability of the Variant Validator pipeline, especially in contexts requiring multiple or iterative queries.

To overcome computational inefficiency, a preprocessing and database integration strategy was designed. Instead of querying the full dbSNP VCF at runtime, all rsIDs and their corresponding genomic coordinates were pre-extracted and stored in a NoSQL database, enabling constant-time lookup operations.

The adopted solution relies on the integration of structured JSON documents into Amazon DocumentDB [53], a fully managed, MongoDB-compatible NoSQL document database service provided by Amazon Web Services (AWS). DocumentDB stores data as flexible JSON-like documents within collections and supports indexed queries for low-latency data retrieval.

Unlike relational databases, DocumentDB does not require a fixed schema and can support dynamic and heterogeneous data. Each record is stored as an independent document, making this model particularly suitable for representing variant records, where each rsID may be associated with multiple genomic representations (e.g., GRCh37 and/or GRCh38 coordinates).

The integration of genomic mappings derived from the complete dbSNP VCF files (GRCh37 and GRCh38 builds) follows a structured preprocessing pipeline:

1. Chromosome-level splitting of dbSNP VCF files

   The original dbSNP VCF files for GRCh37 and GRCh38 are extremely large and computationally expensive to process as single units. Therefore, each file was split by chromosome.

   This operation produced two VCF files (hg19 and hg38) for each chromosome, resulting in a total of 25 chromosome-specific VCF pairs (22 autosomes, chromosome X, chromosome Y and mitochondrial chromosome).

   Splitting the files enabled parallel and memory-efficient processing of smaller and more manageable datasets.

2. Generation of chromosome-specific JSON files

   For each chromosome-specific VCF file, a custom Python script was executed in order to extract and restructure the relevant information. The script iterates over all rsID entries contained in the VCF file, retrieves the corresponding genomic coordinates, and converts them into standardized HGVS genomic notation (g.). For each rsID, the genomic representation(s) both relative to the GRCh37 assembly and GRCh38 assembly are collected when available. The extracted information are then organized into structured JSON documents that associate each rsID with its genome-build–specific HGVS genomic variants.

   For each chromosome, a JSON file was generated containing rsID-to-genomic mappings, as shown in Figure 3.10.

   The genomic variants are expressed using chromosome-based HGVS notation (e.g., `chrY:g.3631610A>T`). Both genome builds may be present for a given rsID, enabling build-aware resolution at query time. Each line of the file corresponds to a single document that is subsequently imported into the DocumentDB collection. Each JSON file therefore represents a complete rsID-to-genomic mapping dataset for a specific chromosome.

3. Loading JSON files into a single DocumentDB collection

```
{"rs": "rs10000002", "37": ["chrY:g.3631610A>T"], "38": ["chrY:g.3763569A>T"]}
{"rs": "rs1000000208", "37": ["chrY:g.8315245G>C"], "38": ["chrY:g.8447204G>C"]}
{"rs": "rs1000001872", "37": ["chrY:g.17891648C>A"], "38": ["chrY:g.15779768C>A"]}
{"rs": "rs1000001998", "37": ["chrY:g.18136039C>T"], "38": ["chrY:g.16024159C>T"]}
{"rs": "rs1000002355", "37": ["chrY:g.19426772G>A"], "38": ["chrY:g.17314892G>A"]}
{"rs": "rs1000002676", "37": ["chrY:g.8485307A>G"], "38": ["chrY:g.8617266A>G"]}
{"rs": "rs1000002837", "37": ["chrY:g.8128161A>C"], "38": ["chrY:g.8260120A>C"]}
```


Figure 3.10: **Example of chromosome-specific JSON records generated from dbSNP VCF files.** Each line corresponds to a single JSON document containing an rsID (field `"rs"`) and its associated genomic HGVS representations for the GRCh37 (field `"37"`) and GRCh38 (field `"38"`) genome assemblies. The genomic variants are expressed in chromosome-based HGVS notation (e.g., `chrY:g.3631610A>T`). Both genome builds may be present for a given rsID, enabling build-aware resolution during query execution.

The 25 generated JSON files were subsequently imported into a single Amazon DocumentDB collection. Although the records were produced from chromosome-specific VCF files during the preprocessing phase, they were ultimately aggregated into a unified dataset within the database.

From the perspective of DocumentDB, the chromosomal origin of each record is not relevant at query time. All rsID documents coexist within the same collection and are accessed through queries performed exclusively on the `rsid` field. By creating an index on this key, the database enables highly efficient lookup operations, with near constant-time retrieval performance (not up to one second) regardless of the original chromosome of the variant.

This procedure eliminates the need for runtime parsing and scanning of large dbSNP VCF files. Instead of repeatedly interrogating large variant datasets, rsID resolution is reduced to a direct database query. As a result, query latency is significantly reduced and the system becomes scalable to millions of rsID lookups without performance degradation.

Furthermore, the integration is fully compatible with the existing `ganno` workflow. When an rsID is provided as input, the system performs a lookup in the DocumentDB collection, retrieves the corresponding genomic HGVS representation for the requested genome build (GRCh37 or GRCh38), and forwards the extracted genomic variant directly to the Variant Validator pipeline.

This boosted Variant Validation module, built upon TransVar, has been named **Variant Validator** and will be referred to as such in the following chapters.

Chapter 4

Validation and benchmarking

4.1 Validation datasets and strategy

The aim of this chapter is to systematically evaluate the Variant Validator, comparing the behavior of the original TransVar implementation with the modified and extended version developed and discussed in the previous chapter. The differences assessed are the coordinate projection, HGVS compliance, and transcript-level mapping, with particular attention to edge cases such as UTR variants.

To ensure a fair and reproducible comparison, both tool versions were executed on the same curated reference datasets.

4.1.1 Benchmark datasets

Two independent datasets were selected to perform the validation analyses.

The first dataset consists of variants derived from the ClinVitae database, including all entries released up to 2017 [55]. ClinVitae is a publicly available repository collecting clinically observed genetic variants both benign and pathogenic. The database aggregates curated variant interpretations performed by clinical experts, from multiple public resources, including ClinVar [56].

Each record includes genomic coordinates (in VCF-like format).

This dataset was selected for benchmarking purposes because of its heterogeneity of variant types, since it includes single nucleotide variants, insertions, deletions, delins and duplications.

This dataset was specifically used to evaluate the forward validation workflow starting from genomic VCF input described in Par. 3.2.2.1. The comparison focused on the mapping results produced by TransVar and the Variant Validator with respect to gene and transcript assignment and c./p. mappings, and these outputs were systematically compared against the annotations generated by the SnpEff annotation tool. A detailed overview of SnpEff has been provided in Par. 2.3.1.

The Clinvitae dataset contains a total of 14,287 variants. All validation analyses were performed using the GRCh37 reference genome assembly. For transcript annotation, both RefSeq and Ensembl transcript databases were employed.

The second benchmark dataset consists of 2,001 enGenome internal manually curated variants formatted as structured records containing multiple possible annotation fields, including gene symbol, transcript accession, rsID, c., p. and mithocondrial genomic-level HGVS (m.). Each entry may contain different combinations of populated fields. Representative examples are shown in Figure 4.1.

```
[
 {‘‘gene’’: ‘‘CFTR’’,  ‘‘rs’’: ‘‘’’, ‘‘hgvs_c’’: ‘‘c.220C>T’’,
  ‘‘hgvs_p’’: ‘‘p.Arg74Trp’’, ‘‘hgvs_m’’: ‘‘’’,
  ‘‘genomic_coord’’: ‘‘’’, ‘‘transcript’: ‘‘’’},

 {‘‘gene’’: ‘‘ARID1A’’,‘‘rs’’: ‘‘’’, ‘‘hgvs_c’’: ‘‘’’,
  ‘‘hgvs_p’’: ‘‘p.Thr294ProfsTer69’’, ‘‘hgvs_m’: ‘‘’’,
  ‘‘genomic_coord’’: ‘‘’’, ‘‘transcript’’: ‘‘’’},

 {‘‘gene’’: ‘‘ATM’’, ‘‘rs’’: ‘‘’’, ‘‘hgvs_c’’: ‘‘c.241A>G’’,
  ‘‘hgvs_p’’: ‘‘p.Asn81Asp’’, ‘‘hgvs_m’’: ‘‘’’,
  ‘‘genomic_coord’’: ‘‘’’, ‘‘transcript’’: ‘‘NM_000051.4’’},

 {‘‘gene’’: ‘‘MT-TK’’, ‘‘rs’’: ‘‘’’, ‘‘hgvs_c’’: ‘‘’’,
  ‘‘hgvs_p’’: ‘‘’’, ‘‘hgvs_m’’: ‘‘m.8344A>G’’,
  ‘‘genomic_coord’’: ‘‘’’, ‘‘transcript’’: ‘‘’’}
]
```

Figure 4.1: **Example of structured variant records used in the second benchmark dataset.** Each entry represents a single variant encoded as a structured record with multiple optional field. Different combinations of populated fields allow validation of multiple TransVar workflows (g. $\rightarrow$ c./p., c. $\rightarrow$ g./p., and p. $\rightarrow$ g./c.).

Unlike the ClinVitae dataset, which was exclusively used to benchmark genomic-driven validation (g. $\rightarrow$ c./p.), this second dataset was designed to also validate the additional input modalities supported by TransVar. Because entries may contain c., p., gene-based, transcript-based, or rsID-based information, they allow systematic testing of all supported projection workflows.

For each of the 2,001 variants, both TransVar and the Variant Validator were executed, and the resulting g./c./p. annotations were collected. The outputs were then compared against the annotations produced by VEP. Unlike SnpEff, which primarily operates on VCF-like genomic inputs, VEP supports flexible HGVS-based inputs at coding and protein levels, making it suitable for benchmarking non-VCF variant representations. A detailed description of VEP and its validation framework has been provided in Par. 2.3.1.

4.1.2 Evaluation workflow

In order to quantitatively compare the original TransVar and the Variant Validator implementations, a structured mismatch classification strategy was defined for each benchmark dataset. The goal was not only to measure concordance rates, but also to categorize discrepancies according to their biological and validation impact. Detailed results are presented in the following sections together with graphical summaries.

ClinVitae dataset (VCF-driven workflow). For the ClinVitae benchmark, the analysis focused exclusively on genomic-driven validation starting from VCF input. The complete set of 14,287 variants was processed in batch mode by both TransVar and SnpEff, which returned, for each input variant, the associated gene(s), overlapping transcript(s), and corresponding c./p. HGVS representations together with predicted effects.

The mismatch classification was performed hierarchically:

1. output generation: first, it was verified whether the tools produced at least one validation record for each VCF input. Variants that did not produce an output were analyzed separately;

2. top-level concordance: for variants with non-null output in both tools, concordance was first evaluated for core genomic attributes, namely chromosome and gene assignment;

3. transcript concordance: the set of transcripts reported by each tool was compared, quantifying how many transcript identifiers were shared or tool-specific;

4. validation-level concordance: on the subset of transcripts common to both TransVar and SnpEff, concordance was assessed at multiple levels: predicted functional effect and HGVS representations at c. and p. levels.

Second benchmark dataset (multi-input workflow). The second dataset required a different classification strategy due to the heterogeneous structure of the input records. Each variant could contain different combinations of gene symbol, transcript accession, rsID, c., p., or mitochondrial HGVS (m.).

To generate valid queries for TransVar, a prioritization logic was applied to determine the most informative combination of fields to be used as input. When both gene symbol and transcript accession were available, the transcript identifier was preferred as the primary prefix, since transcript-based queries allow a more precise coordinate projection.

Similarly, when both c. and p. HGVS descriptions were present, the coding representation was prioritized. In cases where only the gene symbol was available as prefix, the workflow attempted to identify the transcript whose coding coordinate produced the provided protein consequence.

The rsID field was used only as a fallback identifier when no gene, transcript, or HGVS-based coordinates were available. This strategy ensured that the most specific and informative identifiers were used whenever possible while still allowing validation of records with incomplete annotation fields.

The mismatch analysis followed two main steps:

1. output generation: for each generated query, it was necessary to first determine whether TransVar returned a non-null validation. Null outputs were classified separately to capture resolution failures;

2. cross-tool concordance: for variants with non-null outputs, the retrieved information (gene, transcript, c., p., m. and rsID) were compared between TransVar and VEP. Discrepancies were categorized according to the level at which disagreement occurred (gene symbol, transcripts, coordinate projection, or HGVS representation). The detailed discussion of this cross-tool concordance analysis is deferred to Par. 5.2.2, where the same dataset is considered within a real-world application setting.

4.2 TransVar performance analysis

In this section, the baseline performance of the original TransVar implementation is evaluated on both benchmark datasets described in Par. 4.1.1. The ClinVitae dataset is used to assess the genomic-driven workflow (g. $\rightarrow$ c./p.), comparing TransVar outputs against annotations generated by SnpEff. It is important to note that SnpEff is not considered a ground truth reference; rather, it is used as a consistent comparative baseline to identify and systematically analyze differences in annotation behavior between tools.

The second dataset is employed to evaluate multi-input workflows (c., p., gene- and rsID-based), using VEP as reference tool.

This dual benchmarking strategy allows evaluation of TransVar behavior across complementary scenarios, covering both VCF-based genomic validation and HGVS-driven cross-level projections.

Clinvitae dataset performance analysis As a first step, the original TransVar implementation was evaluated using the ClinVitae dataset.

Among the total of 14,287 input variants, 14,283 produced at least one validation record in the TransVar output, while 4 variants did not produce output. In contrast, these same 4 variants were successfully validated by SnpEff.

A closer inspection revealed that all four cases belonged to the *upstream_gene_variant* category (Figure 4.2). These variants are located upstream of the transcription start site (TSS) of the affected transcripts. Although they may still fall within the broader genomic locus of the gene, they do not overlap any transcript-defined region.

This discrepancy reflects a fundamental difference between the annotation strategies adopted by the TransVar and SnpEff. TransVar follows a strictly transcript-centric approach: as explained in Par. 3.2.1, its internal `.transvardb` structure is organized around transcript models and stores only regions explicitly defined within transcripts (e.g., transcript span, exons, CDS, UTR). As a consequence, only variants overlapping these features can be projected across representation levels. Variants falling outside transcript boundaries cannot be mapped and therefore result in null output.

In contrast, SnpEff adopts a gene-level annotation strategy and can classify variants relative to gene context (e.g., upstream or downstream), even in the absence of direct overlap with transcript-defined regions.

```
1:24194788:A:G
X:33357592:T:C
X:33357839:T:G
X:33358200:G:A
```

Figure 4.2: **Upstream variants with no TransVar output.** The four VCF entries that did not produce any validation in TransVar.

To support a systematic benchmarking of TransVar against SnpEff, a unified comparison output was generated in JSON format. For each VCF input variant, TransVar and SnpEff were compared record-by-record by using a unique variant identifier (`KEY`), defined as `CHR:POS:REF:ALT`. The resulting comparison file stores, for each `KEY`, (i) whether the variant is present in the output of each tool; (ii) a comparison of top-level fields; and (iii) a transcript-level comparison structured by transcript presence and fields concordance (iv).

At the top level, the JSON reports a `presence` block indicating whether the variant was validated by each tool, and a `top_fields` block containing match/mismatch labels for core genomic attributes (chromosome and gene identifiers). At transcript level, transcript identifiers are divided into three groups: transcripts found only in TransVar, transcripts found only in SnpEff and transcripts shared by both. For shared transcripts, field-wise comparisons were stored for `effect`, `hgvs_c` and `hgvs_p`, each labeled as `MATCH` or `MISMATCH`.

The structure for a representative variant can be seen in Figure 4.3.

```
{
  "KEY": "10:100177309:G:A",
  "presence": {"TransVar": true, "SnpEff": true},
  "top_fields": {
    "CHR":  {"status": "MATCH", "TransVar": "10", "SnpEff": "10"},
    "GENE": {"status": "MATCH", "TransVar": "HPS1", "SnpEff": "HPS1"}
  },
  "transcripts": {
    "only_in_TransVar": [],
    "only_in_SnpEff": [],
    "both": {
      "ENST00000325103": {
        "effect": {"status": "MATCH",
                   "TransVar": "3-UTRSNV",
                   "SnpEff": "3_prime_UTR_variant"},
        "hgvs_c": {"status": "MATCH",
                   "TransVar": "c.*12C>T",
                   "SnpEff": "c.*12C>T"},
        "hgvs_p": {"status": "MATCH",
                   "TransVar": null,
                   "SnpEff": null}
      }
      ...
    }
  }
}
```

Figure 4.3: **Example structure of a single entry in the comparison TransVar vs SnpEff output file.** Each record is identified by `CHR:POS:REF:ALT` and reports: (i) tool output generation, (ii) top-level field concordance, and (iii) transcript-level comparison grouped into transcripts found only by one tool or shared by both. For shared transcripts, field-wise match/mismatch labels are provided for effect and HGVS projections (iv).

Top-level comparisons on `CHR` and `GENE` did not produce any mismatches across the entire ClinVitae benchmark.

After verifying top-level consistency, the analysis was moved to transcript-level outputs. Generally, a single VCF variant may overlap multiple isoforms, and both TransVar and SnpEff may return several transcript-specific annotations for the same genomic event.

For this reason, the first step of the low-level benchmarking focuses on *transcript presence* rather than the validation content. In this phase, the objective is to quantify whether the two tools select the same set of transcript identifiers for each input variant, independently of whether downstream fields (effect, HGVS c. and p.) are concordant.

To compute a global overview, transcript lists returned for each variant were extracted from the JSON comparison output and grouped according to the `transcripts` section. Transcript identifiers were classified into three mutually exclusive categories:

- MATCH: transcripts reported by *both* tools;
- MISMATCH_TRANSVAR: transcripts reported *only* by TransVar;
- MISMATCH_SNPEFF: transcripts reported *only* by SnpEff.

Counts were then aggregated across all ClinVitae variants by summing transcript occurrences in each category.

A total of 234,128 transcript annotations were evaluated across the ClinVitae benchmark dataset. Among these, 220,847 transcripts were shared between TransVar and SnpEff, 12,919 were reported only by TransVar, and 362 were reported exclusively by SnpEff (Figure 4.4).

While the majority of transcript annotations are shared between tools, two asymmetric mismatch classes emerge. Particular attention was devoted to the 362 transcripts detected exclusively by SnpEff and absent from TransVar output.

A detailed inspection revealed that all transcripts belonging to the *MISMATCH_SNPEFF* category originated from the Ensembl GTF database. This observation suggested that the discrepancy was not caused by validation logic at projection

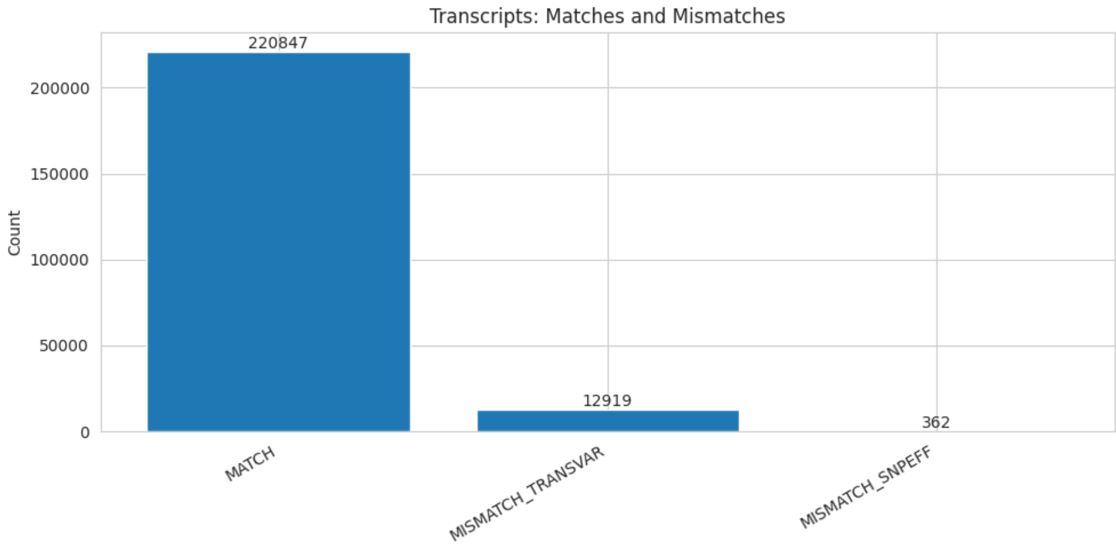

Figure 4.4: **Transcript-level overlap between TransVar and SnpEff across the ClinVitae benchmark.** The bars represent the total number of transcript annotations classified as shared between tools (*MATCH*), reported only by TransVar (*MISMATCH_TRANSVAR*), or reported only by SnpEff (*MISMATCH_SNPEFF*).

level, but rather by differences in transcript database parsing during TransVar preprocessing.

An illustrative example is transcript `ENST00000524880`, which is reported by SnpEff but absent from TransVar internal database. Inspection of the original Ensembl hg19 GTF file showed that the transcript was correctly present in the reference annotation. However, during TransVar preprocessing, transcripts were indexed using `gene_id` as the primary key. In cases where multiple Ensembl `gene_id` values are associated with the same biological gene (i.e., identical `gene_name`), transcripts may be partitioned across distinct internal objects. As a consequence, only transcripts associated with the first encountered `gene_id` were retained, while those belonging to alternative `gene_id` entries under the same `gene_name` were discarded.

This behavior effectively fragmented transcripts belonging to the same gene across separate internal representations and can led to complete loss of specific isoforms. The issue therefore did not reflect a biological disagreement between tools, but a structural limitation in the GTF parsing strategy.

To address this problem, the parsing logic was revised to use `gene_name` as the primary grouping key rather than `gene_id`. This modification ensures that transcripts associated with different Ensembl `gene_id` values but sharing the same `gene_name` are consolidated under a single gene entity. Additionally, only transcripts mapped to official genomic assemblies (NC and MT accessions) were retained, ensuring consistency and avoiding redundancy.

The second category of mismatches, comprising 12,919 transcripts reported only by TransVar, was also investigated. These cases were primarily attributable to transcript biotypes retained in the TransVar Ensembl database but not reported by SnpEff. Specifically, transcripts annotated as *retained_intron* and *nonsense_-mediated_decay* were included in the TransVar internal `.transvardb` files but were not consistently returned by the SnpEff pipeline.

Therefore, the observed *MISMATCH_TRANSVAR* class largely reflects differences in transcript biotype inclusion policies rather than projection errors.

After characterizing transcript overlap, the benchmarking was extended to evaluate validation concordance within the set of shared transcripts. In this step, the analysis were restricted to transcript identifiers present in the `both` group of the JSON comparison output, and concordance was assessed independently for three transcript-level information: `effect`, `hgvs_c`, and `hgvs_p`. Each shared transcript contributes one comparison event per field, resulting in a total of 220,847 evaluations for each field across the ClinVitae benchmark.

The resulting match/mismatch distribution per field is shown in Figure 4.5.

It is important to emphasize that mismatches in the `effect` field must be interpreted cautiously. SnpEff employs a more granular consequence ontology than TransVar, and several apparent discrepancies arise from differences in terminology or resolution rather than from true biological disagreement. For the purposes of this benchmarking, strict lexical identity of effect labels was not considered a primary accuracy criterion. To improve comparability, a mapping between the two consequence ontologies was introduced, allowing harmonization of effect categories and reducing discrepancies due to annotation granularity.

Instead, HGVS coordinate projections (`hgvs_c` and `hgvs_p`), which provide a standardized and structurally explicit representation of the variant relative to

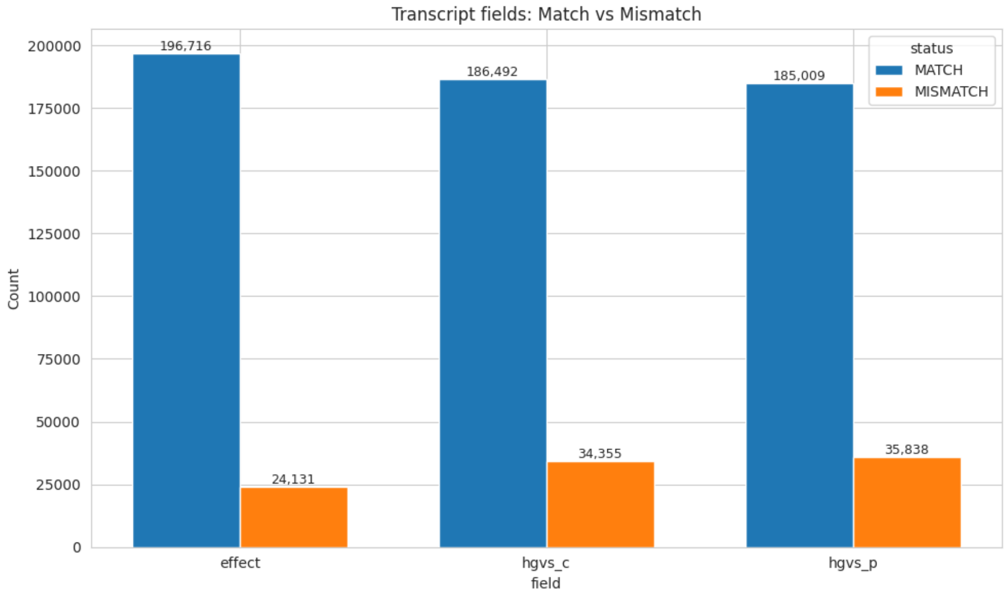

Figure 4.5: **Field-level concordance for transcripts shared between TransVar and SnpEff.** For each transcript identifier reported by both tools, the figure shows the number of `MATCH` and `MISMATCH` outcomes for the fields `effect`, `hgvs_c`, and `hgvs_p`.

the transcript model, constitute the central measure of projection correctness. Accordingly, the subsequent analysis focuses primarily on these two HGVS-level discrepancies.

Interpretation of hgvs_c mismatch categories

Coding-level mismatches (`hgvs_c`) observed among shared transcripts were stratified into four principal categories:

- transvar_only: this category includes all cases in which TransVar reports a transcript-relative coding representation, while SnpEff returns a null value for the same transcript. These mismatches occur for non-coding transcripts. As said in the Par. 3.2.3, TransVar supports the validation of non protein-coding transcripts (`n.` notation) even though it miscorrectly uses the `c.` prefix. However, SnpEff restricts its validation workflow only to protein-coding transcripts. Therefore, this class reflects differences only for this specific transcript biotype;

- different_position: this category captures cases in which both tools report a coding-level HGVS string, but the primary numeric coordinate differs. These discrepancies mainly involve variants located in the 5' untranslated regions. As discussed in the Par. 3.2.3, UTR variants are mapped incorrectly by the original TransVar implementation and the resulting HGVS c. notations are not compliant with canonical HGVS rules;

- UTR_vs_Intronic: this class includes mismatches in which one tool assigns the variant to an intronic coordinate format (using "+" offsets from exon boundaries), while the other expresses it using UTR-specific notation (e.g., "*" numbering for the 3'UTR).

  Importantly, both `Different_position` and `UTR_vs_Intronic` stem from the same underlying issue: differences in UTR-aware mapping, both for 5' and 3';

- other_difference: residual mismatches not falling into the previous categories were grouped under this class. Most cases correspond to INDEL variants for which the two tools differ in representational granularity. While positional interpretation is indeed concordant, TransVar and SnpEff encode the altered sequence with different levels of explicit detail. These discrepancies therefore reflect differences in HGVS reporting style rather than projection errors.

Representative examples of variant mismatches for each category are reported in Table 4.1.

The distribution of coding-level mismatch categories is summarized in Figure 4.6, while Figure 4.7 further stratifies the same categories according to transcript database origin (Ensembl versus RefSeq).

Interpretation of hgvs_p mismatch categories

Protein-level mismatches (`hgvs_p`) observed among shared transcripts were stratified into four principal categories:

- transvar_only: This category includes cases in which TransVar reports a protein-level HGVS representation while SnpEff returns a null value;

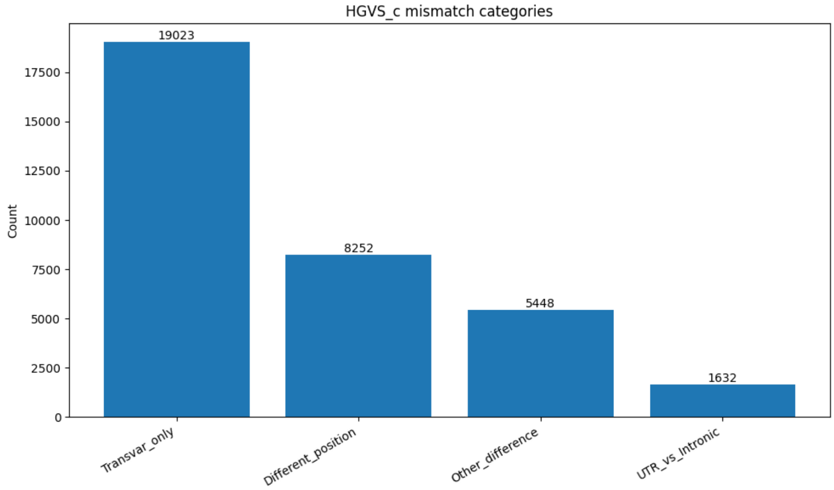

Figure 4.6: **Global distribution of TransVar vs SnpEff HGVS_c mismatch categories.** The bar plot reports the total number of coding-level mismatches classified into the categories *Transvar_only*, *Different_position*, *Other_difference* and *UTR_vs_Intronic* The majority of discrepancies are attributable either to transcript biotype handling differences (*Transvar_only*) or to projection-level differences involving UTR coordinate mapping.

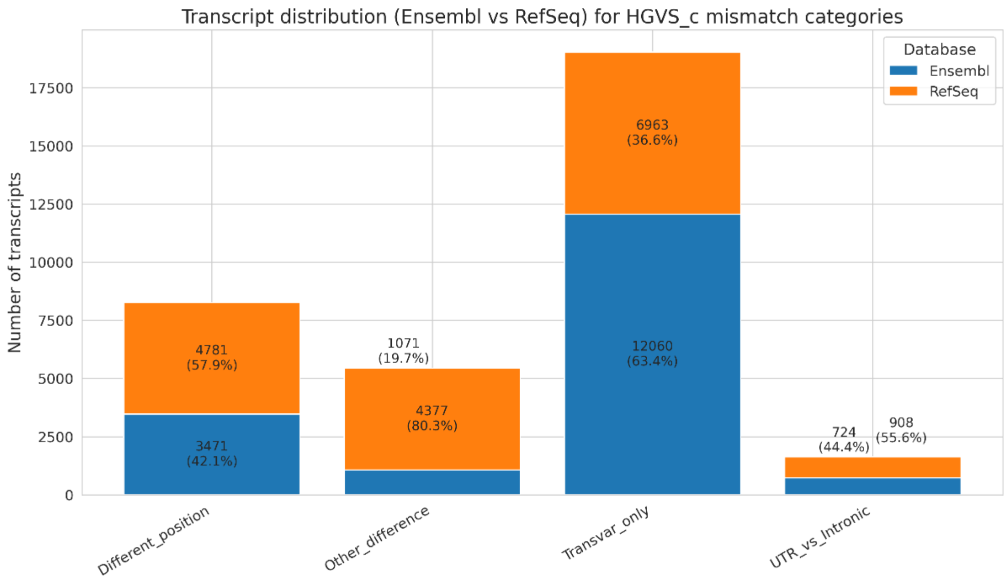

Figure 4.7: **TransVar vs SnpEff HGVS_c mismatch categories stratified by transcript database (Ensembl vs RefSeq).** Each bar is subsplit to show the contribution of Ensembl and RefSeq transcripts within each mismatch category. Percentages indicate the relative contribution of each database within the respective category.

Category	**Input (VCF)**	**Transcript**	**TransVar (c.)**	**SnpEff (c.)**
Transvar_only	10:103372126:A:G	NR_136613.2	c.1380T>C	null
Different_position	10:89692920:T:TA	NM_001304718.2	c.1-19053dupA	c.-346dupA
UTR_vs_Intronic	10:100177309:G:A	NM_000195.5	c.2103+12C>T	c.*12C>T
Other_difference	12:49416388:TTTCTCCCGCCGGTTGGC:TG	NM_003482.4	c.16306_16322delinsC	c.16306_16322delGCCAACCGGCGGGAGAAinsC

Table 4.1: **Examples of TransVar vs SnpEff HGVS_c mismatch categories.** For each category, the genomic input (VCF), transcript accession, and corresponding HGVS_c strings reported by TransVar and SnpEff are shown.

- SnpEff_only: This category includes cases in which SnpEff reports a protein-level consequence while TransVar returns a null `hgvs_p`. A detailed inspection revealed that these mismatches predominantly involve variants classified as *SpliceDonorDeletion* or *SpliceAcceptorDeletion*.

  In the original TransVar implementation, deletions affecting canonical splice sites were filtered out at protein level due to an internal logic block that suppressed `hgvs_p` generation for splice-loss events.

  The underlying rationale was to avoid generating protein-level predictions in cases where disruption of splicing may lead to multiple alternative transcript outcomes (e.g., exon skipping, intron retention, or activation of cryptic splice sites). Because these events can produce different mRNA isoforms, the resulting protein product cannot be uniquely determined, and any single HGVS protein representation would therefore be inherently ambiguous.

  However, this conservative approach resulted in systematic absence of protein-level annotations even when the variant satisfied two critical conditions: (i) both breakpoints were located within exon regions; and (ii) the variant fell entirely within the annotated CDS. In such cases, the coding consequence is in fact determinable under the reference transcript model.

To address this limitation, the splice-related suppression logic was revised. The updated implementation now allows `hgvs_p` computation for splice-site variants whenever the affected region is exonic and contained within the CDS. As a result, variants impacting splice donor or acceptor sites but structurally confined to coding exons can now generate a consistent protein-level HGVS annotation.

- different_position: This category captures cases in which both tools report a protein-level HGVS string, but the amino acid position differs. A detailed inspection revealed that the vast majority of transcripts belonging to this class are characterized by incomplete coding sequences (CDS), either at the 5' end, the 3' end, or both.

  An incomplete CDS indicates that the annotated transcript does not contain a fully defined coding region, meaning that the start codon and/or the stop codon are missing or not reliably annotated within the reference assembly. As a consequence, the resulting protein sequence is not anchored to a stable and biologically complete reference.

  When the CDS is incomplete at the 5' end, the N-terminal portion of the protein may be truncated or undefined; when incomplete at the 3' end, the annotated stop codon may be missing or shifted. In both situations, the inferred protein sequence becomes unstable and may differ between tools depending on how these incomplete boundaries are handled.

  This structural instability propagates directly to amino acid numbering, leading to systematic shifts in reported positions even when the underlying genomic variant is identical. Importantly, CDS boundary incompleteness does not affect only the protein-level projection (`hgvs_p`), but also influences the coding-level coordinate system (`hgvs_c`) and, indirectly, the genomic projection (`hgvs_g`).

  Therefore, most `Different_position` mismatches at protein level (but also at cDNA level) are not caused by erroneous variant projection, but by these inconsistencies in CDS boundary completeness. This phenomenon is particularly prevalent among Ensembl transcripts, where partial CDS annotations

(5' and/or 3' incomplete coding sequences) are more frequently observed compared to RefSeq entries.

To address this issue, CDS completeness flags were extracted from the Ensembl GTF annotation files (specifically the `cds_start_NF` and `cds_end_NF` attributes). These flags indicate whether the coding sequence is incomplete at the 5' or 3' end. Such information was incorporated into the transcript preprocessing step so that, whenever a transcript with incomplete CDS is validated (in any of the three supported TransVar workflows: g→c/p, c→g/p, or p→g/c), a dedicated flag is emitted to explicitly indicate CDS boundary incompleteness. This allows downstream interpretation of protein-level mismatches to distinguish true projection discrepancies from artifacts related to incomplete transcript models.

- other_difference: The remaining mismatches were grouped under this category. The majority of these cases correspond to frameshift variants. In such events, TransVar produces a fully HGVS-compliant protein-level description, explicitly reporting both the frameshift and the position of the newly generated termination codon, including the number of amino acids formed before the new stop signal. In contrast, SnpEff reports only a simplified frameshift label indicating "fs" without specifying the position of the newly formed terminator codon.

Representative examples of variant mismatches for these categories are shown in Table 4.2.

The distribution of protein-level mismatch categories is defined in Figure 4.8, while Figure 4.9 stratifies the same categories according to transcript database origin (Ensembl versus RefSeq).

Second benchmark dataset performance analysis The baseline behavior of the original TransVar implementation was next evaluated on the second benchmark dataset, which comprises 2,001 structured variant records and supports heterogeneous input modalities. In this setting, TransVar outputs were systematically compared against annotations generated by the Ensembl Variant Effect Predictor (VEP), which serves as the reference tool for HGVS-based input handling.

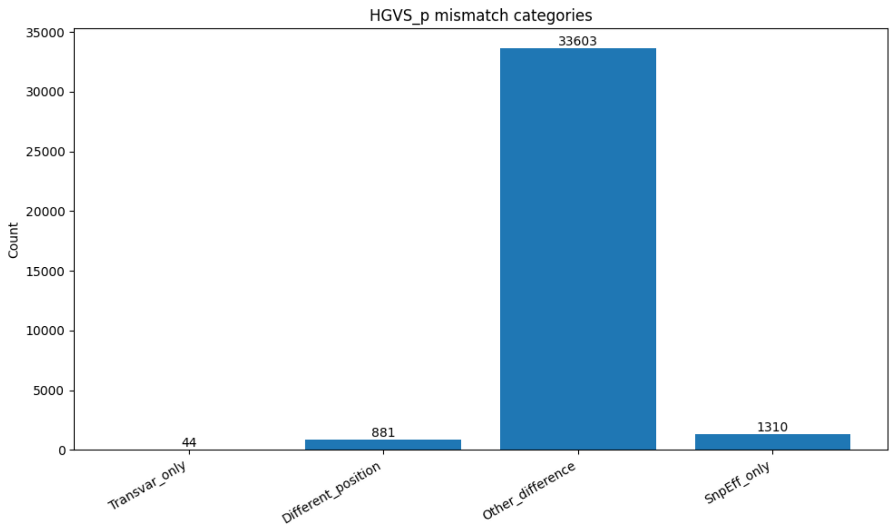

Figure 4.8: **Global distribution of TransVar vs SnpEff HGVS_p mismatch categories.** Total number of protein-level mismatches classified as *Transvar_only*, *Different_position*, *Other_difference*, and *SnpEff_only*. Most discrepancies are frameshift representation differences (*Other_difference*) and CDS boundary incompleteness affecting amino acid numbering (*Different_position*).

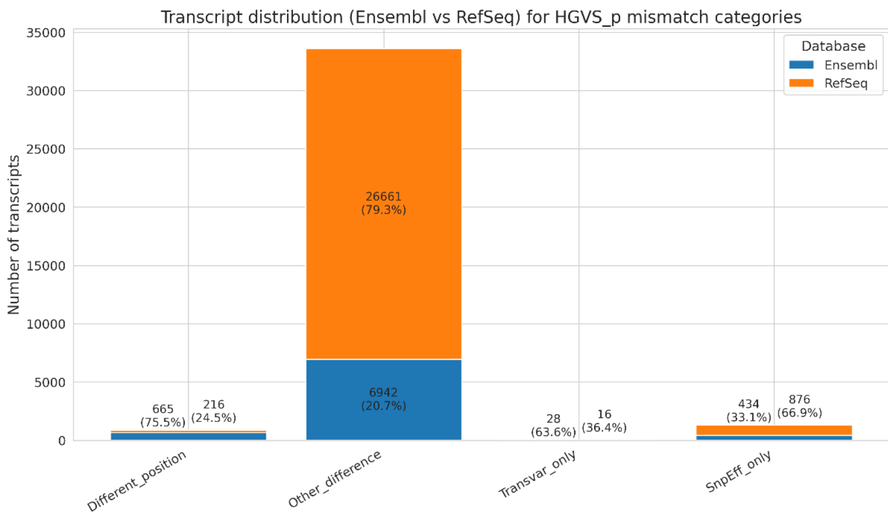

Figure 4.9: **TransVar vs SnpEff HGVS_p mismatch categories stratified by transcript database (Ensembl vs Refseq).** Stacked bars show the relative contribution of Ensembl and RefSeq transcripts within each mismatch category. *Different_position* mismatches are enriched in Ensembl transcripts, consistent with the higher prevalence of incomplete CDS annotations.

Category	**Input (VCF)**	**Transcript**	**TransVar (p.)**	**SnpEff (p.)**
Transvar_only	12:52307550:TG:T	ENST00000388922	p.D176Tfs*82	null
Different_position	10:121429647:A:AGCG	ENST00000450186	p.A102dup	p.A101dup
SnpEff_only	11:31811483:T:A	ENST00000379111	null	p.*423L
Other_difference	10:89720804:ACTTT:A	ENST00000371953	p.T319Kfs*24	p.T319Kfs

Table 4.2: **Examples of TransVar vs SnpEff HGVS_p mismatch categories.** For each mismatch class, the genomic input (VCF), transcript accession, and corresponding protein-level HGVS strings reported by TransVar and SnpEff are shown.

Since each record may contain multiple annotation fields (e.g., gene symbol, transcript accession, rsID, HGVS c./p./m., or genomic coordinates), a single query was generated for each entry according to the prioritization logic described in Par. 4.1.2. This strategy selects the most informative available representation to construct a valid TransVar query, ensuring consistent and reproducible validation across the entire dataset.

A primary outcome of this analysis was the identification of a systematic limitation affecting untranslated region (UTR) variants. In particular, these UTR events were provided in this second benchmark as coding-level HGVS expressions (c.). VEP consistently returned a valid cross-level projection (c. $\leftrightarrow$ g.) under the selected transcript model, whereas the original TransVar implementation failed to resolve the variant and returned null outputs.

Across the full benchmark, a total of 60 UTR-related queries produced null output in TransVar despite being successfully resolved by VEP, all of which concerned the cDNA notation.

In addition to these UTR-specific failures, 45 further cases resulted in null output in both TransVar and VEP. Importantly, these shared null outputs did not concern UTR variants, but spanned different categories. Because the failure is reproduced by the external reference tool, these instances were further analyzed and later interpreted as input-level resolution failures (e.g., insufficiently specified transcript context, inconsistent HGVS strings, or records not mappable to the selected reference annotation), rather than as projection errors attributable to TransVar.

To clarify the interpretation of null outputs, two situations have been distinguished. Cases in which TransVar returned a null output while VEP successfully resolved the variant and produced a genomic projection have been defined as false negatives (FN) Conversely, true negatives (TN) correspond to cases in which both tools returned null outputs, indicating that the query could not be resolved given the available input information.

Representative examples of both situations are shown in Table 4.3.

Category	**Input**	**VEP output (g.)**	**TransVar output (g.)**
False Negative	`ARMC5:c.-232A>C`	`chr16:g.31470614A>C`	`null`
False Negative	`ATRIP:c.*1329_*1375del`	`chr3:g.48508282_48508328del`	`null`
False Negative	`NM_001377265.1:c.-17-19975G>A`	`chr17:g.44019712G>A`	`null`
False Negative	`DYNC2LI1:c.*16-1362A>C`	`chr2:g.44053163A>C`	`null`
True Negative	`NBN:p.Ter745_splice`	`null`	`null`

Table 4.3: **Examples of TransVar vs SnpEff output classification in the second benchmark dataset.** False negatives correspond to cases where TransVar fails to resolve the variant while VEP successfully returns a genomic projection. True negatives represent queries that cannot be resolved by either tool, typically due to incomplete or inconsistent input specifications.

4.3 Variant Validator performance analysis

Clinvitae dataset performance analysis Following the identification of the main sources of discrepancies in the baseline analysis, the Variant Validator was evaluated again using the same ClinVitae benchmark dataset. The comparison workflow and evaluation strategy remained identical to the one described in the previous section in order to ensure direct comparability between the baseline and the improved implementation.

Field-level concordance for transcripts shared between the Variant Validator and SnpEff is shown in Figure 4.10.

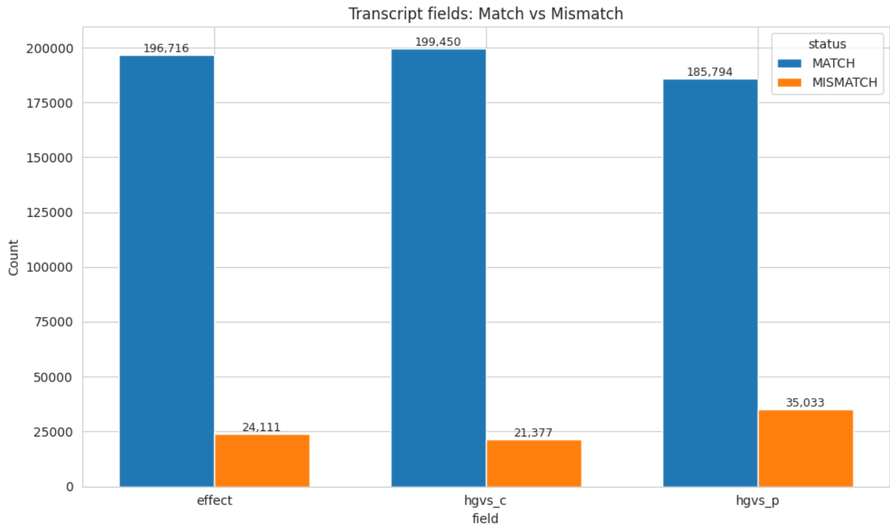

Figure 4.10: **Field-level concordance for transcripts shared between the Variant Validator and SnpEff.** Match and mismatch counts for the fields `effect`, `hgvs_c`, and `hgvs_p` considering transcripts shared between the Variant Validator and SnpEff.

After confirming the consistency of transcript grouping, the analysis was again focused on discrepancies involving HGVS coordinate projections. Both coding-level (`hgvs_c`) and protein-level (`hgvs_p`) mismatches were therefore reclassified using the same categories introduced in the baseline evaluation.

The updated distribution of cDNA mismatch categories is shown in Figure 4.11 while Figure 4.12 shows the stratification of those categories according to transcript database origin.

A major improvement at the coding level concerns the *UTR_vs_Intronic* and *Different_position* categories, which are now substantially reduced compared to the TransVar baseline implementation. This confirms that the the revised coordinate handling with UTR recognition described in Par. 3.2.3 successfully resolves the incorrect projection of UTR variants that previously resulted in intronic-style HGVS coordinates.

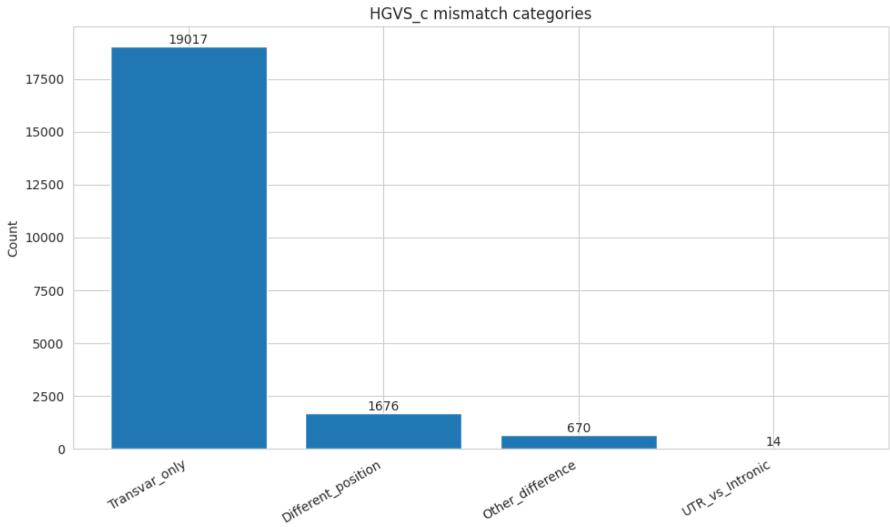

Figure 4.11: **Global distribution of the Variant Validator vs SnpEff HGVS_c mismatch categories** Distribution of coding-level mismatches between the Variant Validator and SnpEff following the modifications introduced in the transcript preprocessing and HGVS generation logic.

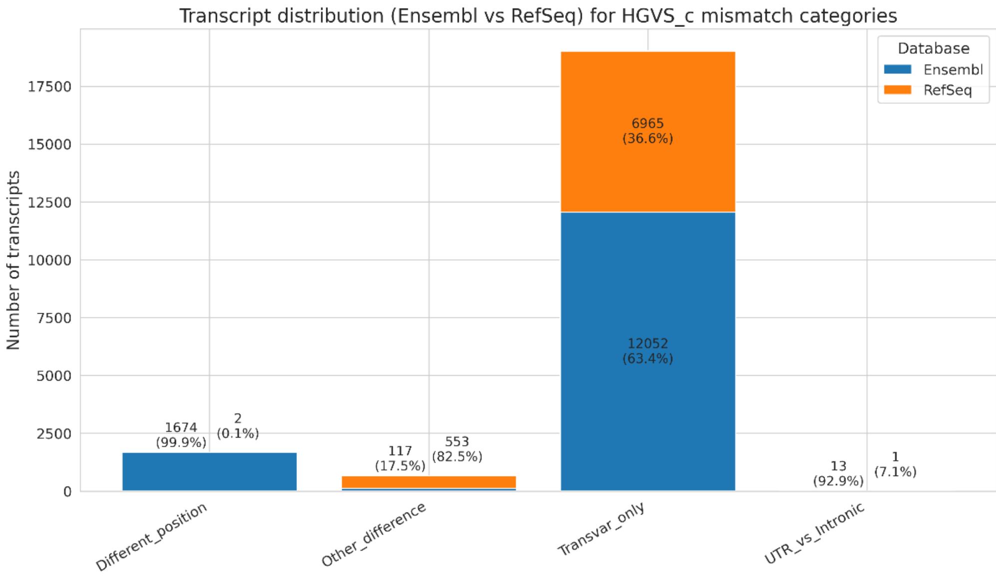

Figure 4.12: **Variant Validator vs SnpEff HGVS_c mismatch categories stratified by transcript database (Ensembl vs RefSeq).** The stacked bars show the relative contribution of Ensembl and RefSeq transcripts within each mismatch category. The remaining *Different_position* discrepancies are almost exclusively associated with Ensembl transcripts characterized by incomplete CDS annotations, while UTR-related mismatches are largely resolved.

Furthermore, the *Different_position* category is now largely restricted to transcripts originating from the Ensembl database that present incomplete CDS annotations. As discussed in Par. 4.2, incomplete CDS boundaries lead to unstable reference coding sequences and consequently to shifts in transcript-relative coordinate numbering. These mismatches therefore do not reflect projection errors but rather structural inconsistencies in transcript models.

The *Other_difference* category also shows a substantial reduction compared to the TransVar baseline implementation. This improvement results from modifications introduced in the HGVS generation logic for insertion and deletion events. The Variant Validator implementation explicitly reports the deleted sequence in the INDELs representation, aligning its output format more closely with the representation adopted by SnpEff.

The few remaining discrepancies within this category correspond to alternative normalization choices for INDEL variants. In particular, the Variant Validator and SnpEff may apply different left-alignment strategies when representing equivalent sequence changes. These differences are largely cosmetic and remain HGVS-compliant. Moreover, the Variant Validator internally reports both the normalized (left-aligned) and the unaligned representations of the variant. As a consequence, even in cases where the primary representation differs from SnpEff, the alternative representation provided by the Variant Validator often corresponds exactly to the HGVS string reported by SnpEff.

After evaluating the improvements at coding level, the analysis was extended to protein-level annotations. The updated distribution of protein-level mismatch categories is shown in Figure 4.13. Figure 4.14 further stratifies these categories according to transcript database origin.

At protein level, the only category directly addressed by the modifications corresponds to the *SnpEff_only* class. As described in Section 4.2, these mismatches originated from an internal filtering rule in the original TransVar implementation that suppressed protein-level annotations for variants affecting canonical splice sites.

The developed Variant Validator revises this logic by allowing protein-level HGVS generation for splice-site variants whenever the affected region lies within exonic coordinates and is contained within the CDS. As a result, splice-related

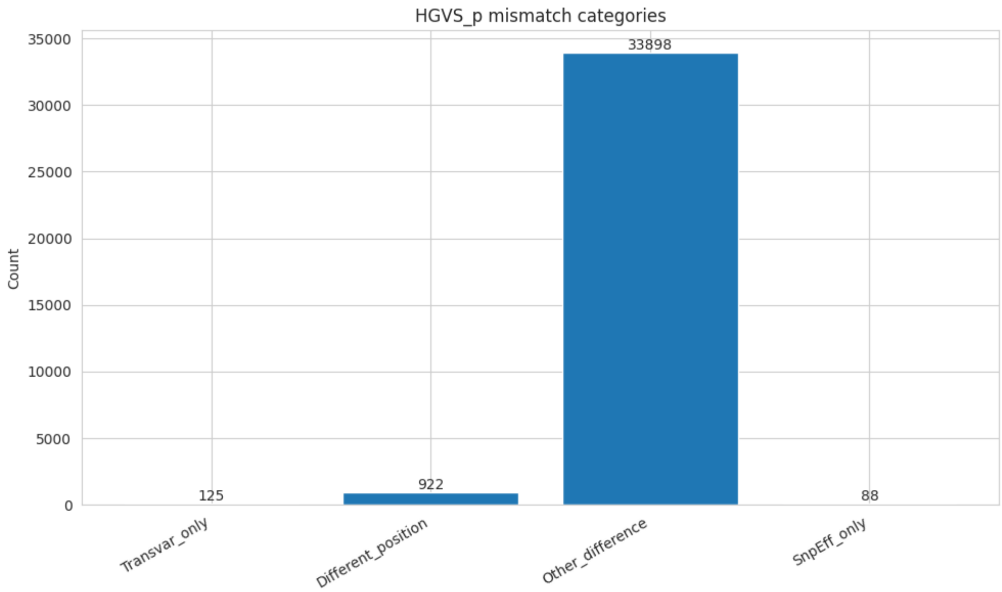

Figure 4.13: **Global distribution of the Variant Validator vs SnpEff HGVS_p mismatch categories.** The *SnpEff_only* class is substantially reduced relative to the baseline analysis due to the updated handling of splice-site variants within coding regions.

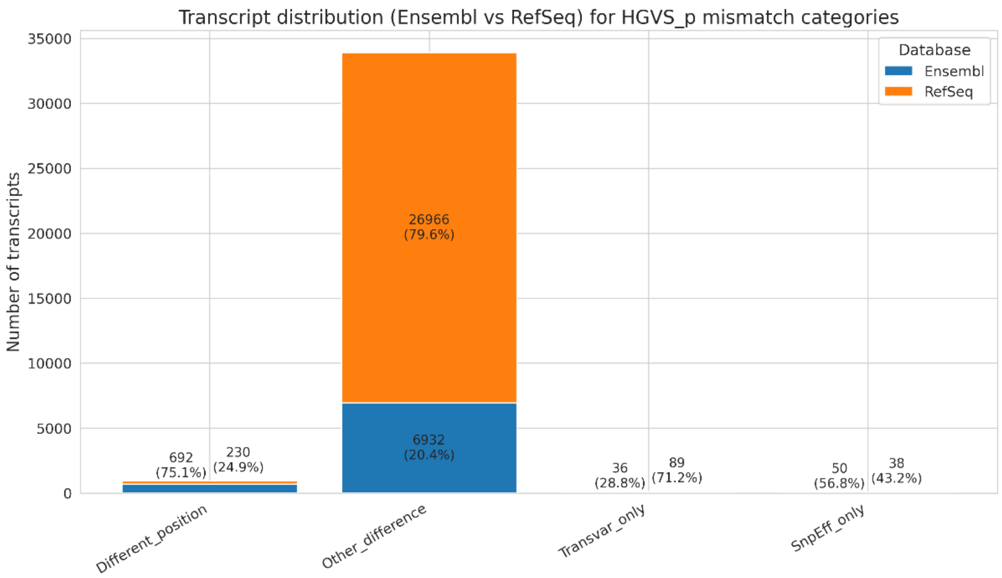

Figure 4.14: **Variant Validator vs SnpEff HGVS_p mismatch categories stratified by transcript database (Ensembl vs Refseq).**

variants that previously produced null protein annotations with TransVar now generate consistent `hgvs_p` outputs, substantially reducing the *SnpEff_only* mismatch category.

Most residual discrepancies involve frameshift variants, for which SnpEff often reports simplified protein consequences, whereas the Variant Validator produces fully HGVS-compliant annotations. Additional cases within the *Different_position* category are largely associated with Ensembl transcripts characterized by incomplete CDS boundaries. Both cases have been explained in Section 4.2.

Second benchmark dataset performance analysis After implementing the modifications to the c. $\rightarrow$ g. projection workflow described in Par. 3.2.3, the second benchmark dataset was processed again using the developed Variant Validator.

The revised mapping strategy successfully resolved all previously identified false negative cases. In particular, the improved handling of UTR coordinates and transcript-aware projection logic allowed the Variant Validator to correctly map coding-level HGVS expressions that previously produced null outputs. As a result, all 60 UTR-related queries that were unresolved in the TransVar baseline implementation are now projected to the correct genomic coordinates.

The only remaining 45 null outputs correspond to the previously identified true negative cases, in which both the Variant Validator and VEP fail to resolve the input query. As discussed in the previous section, these cases are attributable to incomplete or inconsistent input information rather than limitations of the variant projection framework.

A detailed analysis of cross-tool concordance between the Variant Validator and VEP on the second benchmark dataset is presented in Chapter 5, where the same benchmark is further examined in the context of a real-world application scenario.

Chapter 5

Application in a Real-World Use Case: A variant validation framework for VarChat

5.1 VarChat pipeline and possible integration of the Variant Validator

The Variant Validator described and benchmarked in the previous chapters provides a robust framework for verifying and standardizing variant representations across genomic, coding, and protein coordinate systems. Beyond ensuring HGVS-compliant notation and consistent coordinate projections, such functionality can support applications designed to assist genomic variant interpretation.

One example is VarChat [57], a generative AI–based platform developed by enGenome srl that enables users to retrieve and summarize the scientific literature associated with a specific genetic variant. Through a conversational interface, users can submit queries describing a variant using HGVS notation together with the gene symbol or the transcript description (both Ensembl and RefSeq are allowed), or alternatively by providing a dbSNP identifier or the VCF-format nomenclature. VarChat then searches the biomedical literature and generates concise summaries describing biological aspects and the clinical relevance of the queried variant, together with references to the most relevant and up-to-date scientific publications

[58].

However, user-provided variant inputs may be incomplete or may even correspond to non-existent variants because of semantic inaccuracies or biological inconsistencies. In such cases, literature retrieval may become inefficient, potentially leading to incorrect information reported in the variant summary.

To address this limitation, the variant validation framework developed in this work can be integrated into the VarChat pipeline as a preprocessing step. The Variant Validator first verifies the correctness of the user-provided variant representation, preventing the system from initiating searches for invalid or biologically inconsistent variants. Once a valid representation is confirmed, it also generates the equivalent variant descriptions across different coordinate systems (genomic, transcript, protein, and rsID).

By enriching the variant representation with additional standardized descriptors, the system can formulate a greater number of precise literature search queries, one for each description, thus improving both the efficiency of the retrieval process and the relevance of the information returned.

The following section provides an overview of the actual VarChat platform and its main components.

Overview of the VarChat Platform

An overview of the VarChat platform and its user interaction interface is shown in Figure 5.1. The interaction begins when the user submits a query containing a genetic variant expressed using HGVS nomenclature together with the corresponding gene symbol, or alternatively a dbSNP identifier (A).

Once the query is received, VarChat processes the requested variant and searches the biomedical literature for publications mentioning the variant (B–C). Retrieved articles are ranked according to relevance and used as contextual information to generate a concise description of the variant and its potential biological or clinical implications.

Alongside the generated summary, the platform displays the most relevant references associated with the queried variant (D). When available, up to fifteen publications are presented, sorted by relevance, with direct links to PubMed and an indication of the total number of retrieved papers. Moreover, if the variant is present in ClinVar, the corresponding records with the associated condition, clinical significance, and review status are provided.

Finally, VarChat provides a feedback mechanism that allows users to evaluate the generated response through a rating system and optionally provide comments (E).

Internal VarChat processing pipeline and possible integration of the Variant Validator

The internal VarChat processing pipeline from user input to summary generation is illustrated in Figure 5.2.

The process begins with the user submitting a query containing a genomic variant (Figure 5.2 A.). The input may include a gene symbol or a transcript description together with a variant expressed using HGVS nomenclature, a dbSNP identifier, or genomic coordinates. The query is then processed through a Named Entity Recognition (NER) module (Figure 5.2 B.), which extracts the relevant biological entities from the user prompt [59].

Once the relevant entities have been identified, the system attempts to standardize the variant representation and obtain additional annotations (Figure 5.2 C.). In the current VarChat implementation, this step relies on external validation services through API calls to the Ensembl Variant Effect Predictor (VEP). Through this service, VarChat can compute standardized HGVS representations or convert identifiers such as dbSNP IDs into equivalent variant descriptions.

VEP is a widely adopted tool for variant validation and coordinate translation. As discussed in Par. 2.3.1, it supports HGVS-based inputs and is capable of performing cross-level coordinate mapping between genomic (g.), transcript (c.), and protein (p.) representations, similarly to TransVar.

However, these functionalities are mainly accessible through Ensembl web services or REST API interfaces rather than through a fully self-contained local validation engine. As a consequence, VarChat depends on external services to perform variant normalization, coordinate translation, and variant autocompletion.

This architectural dependency introduces operational limitations. In particular, when the VEP API service is temporarily unavailable or experiencing downtime, the normalization and autocompletion steps cannot be executed. Under

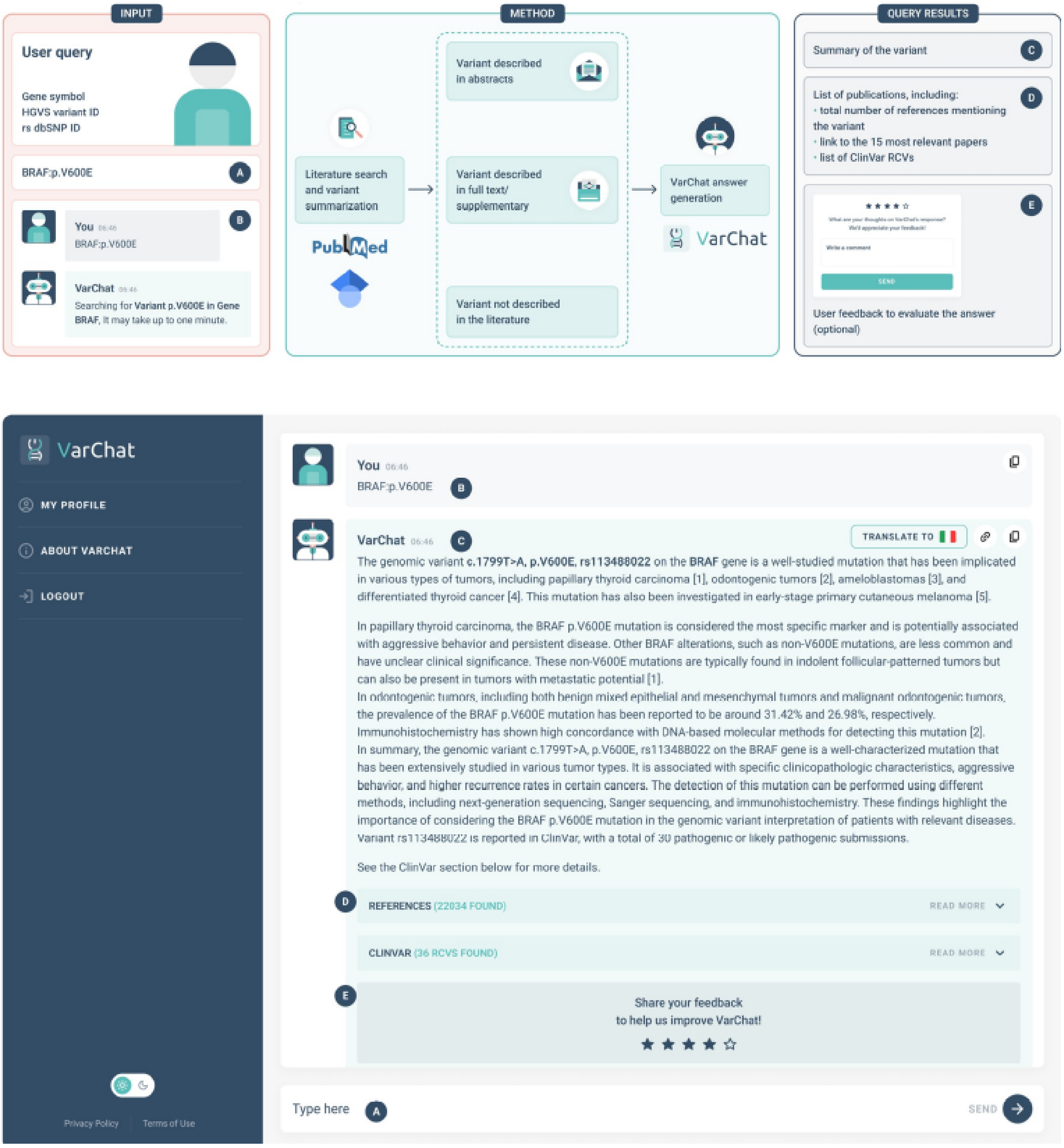

Figure 5.1: **VarChat platform overview.** (A) User prompt enabling variant queries using HGVS nomenclature together with the gene symbol. (B) Processing of the queried variant by the VarChat system. (C) Retrieval of scientific literature mentioning the variant and generation of a summarized description. (D) Display of the most relevant references (up to 15), sorted by relevance, with direct links to PubMed and indication of the total number of retrieved publications. (E) Feedback system allowing users to evaluate the generated response through a 5-star rating and optional comments [57].

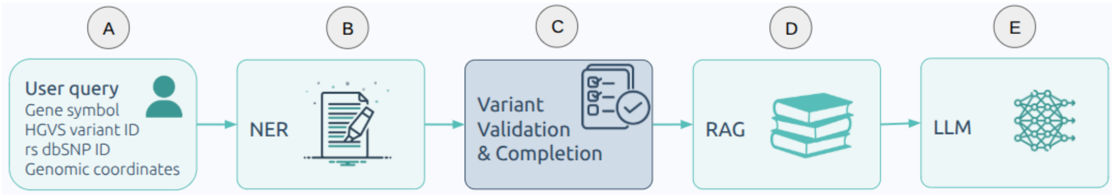

Figure 5.2: **VarChat processing pipeline.** The system processes a user query containing a genomic variant through a sequence of computational steps. (A) The user submits a query including a gene symbol together with a variant identifier, which may be expressed using HGVS notation, a dbSNP identifier, or genomic coordinates. (B) A Named Entity Recognition (NER) module extracts the relevant biological entities from the query, such as the gene name and the variant description. (C) The variant is then validated and standardized to obtain consistent representations across coordinate systems. (D) The normalized variant is used in a Retrieval-Augmented Generation (RAG) step to retrieve scientific publications mentioning the variant from biomedical literature databases. (E) A Large Language Model (LLM) processes the retrieved context and generates a concise summary describing the biological and clinical relevance of the queried variant, which is presented to the user together with relevant references.

these circumstances, the pipeline is unable to validate the user input.

For this reason, integrating the Variant Validator developed in this work into the VarChat pipeline can provide a more robust alternative. By performing variant validation and cross-level mapping locally, the Variant Validator removes the dependency on external validation services and ensures that normalization and autocompletion steps remain always available even when external APIs are not accessible.

After the variant representation is obtained, VarChat performs a Retrieval-Augmented Generation (RAG) step [60], where scientific publications mentioning the queried variant are retrieved from biomedical literature databases (Figure 5.2 D.). The retrieved documents provide contextual information that is used to generate the final summary.

The Large Language Model (LLM) then processes both the user query and the retrieved literature context to produce a concise textual explanation of the variant, summarizing its known biological and clinical implications (Figure 5.2 E.). The final output presented to the user consists of the generated summary together with a ranked list of the most relevant and up-to-date scientific references.

5.2 Validation of user inputs

5.2.1 Input prioritization strategy

The objective of this section is to evaluate the behavior of the Variant Validator in a realistic application scenario, namely its integration within the VarChat pipeline. In particular, the focus is not only on validation performance, but also on the ability of the system to complete and enrich user-provided variant inputs, which is a key requirement for downstream literature retrieval.

To this end, the evaluation is conducted in a setting that closely resembles the operational conditions of VarChat, where user queries are often incomplete and heterogeneous. The Variant Validator is therefore assessed in terms of its capability to reconstruct valid inputs, perform cross-level mapping, and automatically complete missing annotation fields.

The evaluation presented in this section builds upon the dataset of 2001 manually curated variants previously introduced in Par. 4.1.1. While the previous chapter focused on assessing validation performance and null-output categories, here the same dataset is used to evaluate how different variant validation strategies affect the amount of information available for downstream processing within the VarChat pipeline.

Because the variants in the dataset are provided in a trimmed format with some fields potentially missing (i.e., gene symbol and coding description, with missing protein description), a preprocessing stage was implemented to reconstruct a valid input representation before invoking the Variant Validator. This stage identifies which fields are available for each variant and applies a prioritization strategy to determine the most informative representation to use as input for the validation tool.

The reconstructed input is derived from the available fields among: gene symbol, transcript identifier, rsID, HGVS c., p., and m., and genomic coordinates (both g. and VCF-like format). Depending on which fields are present, different strategies are applied in order to generate the most reliable input representation for the Variant Validator.

The prioritization follows three main principles:

- genomic coordinates are always prioritized when available, as they provide the most unambiguous representation of the variant;
- when transcript-level information is available, transcript identifiers are preferred over gene symbols because they uniquely define the reference sequence;
- coding-level descriptions (*c.*) are prioritized over protein-level descriptions (*p.*) because they retain a direct mapping to genomic coordinates.

Special cases are also handled explicitly. Variants containing only a gene symbol are processed as gene-level queries. However, due to the absence of positional information, no variant validation or coordinate mapping can be performed. Therefore, these cases are outside the scope of the validation analysis presented in this work.

If both gene symbol and rsID are available but no positional HGVS representation is provided, the rsID is used as the primary input and subsequently resolved in the Variant Validator to genomic coordinates thanks to the workflow implemented and described in Par. 3.2.4 for rsID input handling.

Once the input representation is reconstructed, the appropriate transcript database is used for Variant Validator calls. RefSeq databases are selected when transcript identifiers correspond to NM accessions, whereas Ensembl databases are used for ENST identifiers. In cases where only a gene symbol is provided together with an HGVS description, the Variant Validator first attempts validation using RefSeq transcripts and then falls back to Ensembl transcripts if no valid result is obtained.

Depending on the reconstructed input, different Variant Validator commands are executed. Genomic variants are processed using the `ganno` command, while transcript-level variants are analyzed using `canno` or `panno`.

Multiple attempts may be required when several database and genome assembly combinations are possible. In particular, transcript-based queries prioritize the GRCh38 assembly and fall back to GRCh37 if no valid annotation is obtained. When gene-level HGVS descriptions are provided, the system first queries RefSeq transcripts and then Ensembl transcripts across the same assembly priority.

For each analyzed variant, the complete output returned by the Variant Validator is stored in a structured JSON file together with metadata describing the database and genome assembly used to obtain the result. This structured representation allows the workflow to preserve both the full set of annotations returned by the Variant Validator and a prioritized representation of the variant that is used for downstream processing within the VarChat pipeline.

Indeed, for input representations that are less specific,such as variants provided only with a gene symbol or genomic coordinates, the annotation process may return multiple transcripts corresponding to the same genomic event. In these cases, all transcript-level annotations returned by the Variant Validator are preserved in the JSON output under the `alternative_transcripts` field. For each transcript, the complete mapping across genomic, coding, and eventually protein coordinate systems (g./c./p.) is stored together with the associated annotation information (e.g., predicted consequence and transcript metadata), ensuring that the full multi-level representation of the variant is retained.

to provide a single representation of the variant, the Variant Validator selects a transcript that is stored in the output JSON `prioritized_variants` field. The prioritization of transcripts follows the MANE framework developed and described in Par 3.2.3. If a MANE Select transcript is present among the mapped transcripts, it is selected as the prioritized representation. If no MANE Select transcript is available, the system prioritizes a MANE Plus Clinical transcript when present; otherwise, a protein-coding transcript among the available annotations is selected.

In the specific case of intergenic genomic variants, where the variant does not fall within a single gene but may be associated with multiple nearby genes, the Variant Validation workflow stores one prioritized transcript for each gene according to the prioritization hierarchy described above. This approach ensures that each potentially relevant gene is represented while maintaining a prioritized transcript selection aligned with clinically curated transcript standards.

In addition to the annotation itself, the workflow performs a set of *coherence checks* between the information provided by the user and the information inferred from the Variant Validator annotation.

These checks evaluate the consistency of the following fields:

- gene: verifies that the gene returned by the Variant Validator matches the gene specified by the user, if present;

- transcript: checks whether the annotated transcript corresponds to the transcript identifier provided in the input, if present;
- hgvs_c: compares the retrieved coding HGVS representation with the one provided by the user;
- hgvs_p: verifies that the predicted protein consequence matches the protein-level HGVS description provided by the user.

Whenever inconsistencies are detected between the user-provided query fields and the annotation returned by the Variant Validator, the corresponding entry in the `coherence_checks` structure is flagged. These flags allow the system to explicitly report mismatches and highlight potential inconsistencies between the original user input and the validated annotation.

Table 5.1 summarizes the main input reconstruction scenarios and the corresponding Variant Validator calls generated by the prioritization strategy.

Example case 1: Gene + HGVS_c + HGVS_p with no incoherences

A representative example of the prioritization strategy occurs when the user provides a gene symbol together with both coding and protein HGVS descriptions. In this scenario, the workflow prioritizes the coding-level representation and performs the initial Variant Validator query using the gene and the *c.* description. The protein-level description is then used as an additional consistency check to ensure that the predicted amino acid change matches the one provided by the user.

For example, considering the following input record extracted from the dataset:

```
{''gene'': ''CFTR'',
 ''rs'': '''',
 ''hgvs_c'': ''c.220C>T'',
 ''hgvs_p'': ''p.Arg74Trp'',
 ''hgvs_m'': '''',
 ''genomic_coord'': '''',
 ''transcript'': ''''}
```

Table 5.1: Input reconstruction and prioritization strategy used to generate Variant Validator queries from trimmed variant records.

User input query	Input reconstruction strategy	Variant Validator command
Gene only	Insufficient positional information	No analysis performed
Gene + rsID	Resolve rsID to genomic coordinates (via DocumentDB) and analyze each genomic alternative	*ganno*
Genomic coordinates	Directly used as primary input	*ganno*
Transcript + HGVS_c	Transcript-level representation prioritized	*canno*
Transcript + HGVS_c + HGVS_p	Query performed using *c.* notation; check for the user-provided *p.* change	*canno*
Gene + HGVS_c	Attempt RefSeq transcripts first, then Ensembl if no valid result is obtained	*canno*
Gene + HGVS_c + HGVS_p	Transcripts filtered to match the user-provided *p.* consequence. Attempt RefSeq transcripts first, then Ensembl if no valid result is obtained	*canno*
Gene + Transcript + HGVS_c/p	Transcript prioritized; gene consistency checked. If both *c.* and *p.* are present, the *p.* consequence is validated against the predicted effect	*canno*
Gene/Transcript + HGVS_p only	If only the gene is provided: attempt RefSeq transcripts first, then Ensembl if no valid result is obtained	*panno*
rsID with multiple genomic loci	All genomic alternatives analyzed separately and stored as independent results	*ganno*

Since both *c.* and *p.* descriptions are available, the reconstructed query becomes:

`CFTR:c.220C>T`

The Variant Validator then attempts validation across the available transcript databases, prioritizing RefSeq transcripts on GRCh38, followed by GRCh37 if no valid result is obtained previously. If necessary, Ensembl transcripts are subsequently explored using the same assembly priority.

For each candidate transcript returned by Variant Validator, the predicted protein consequence is compared with the user-provided protein change (`p.Arg74Trp`). Only transcripts producing a consistent coding and protein-level effect are shown in the final output.

An example of the resulting structured output generated by the Variant Validator is shown in Figure 5.3. For clarity, the figure reports only the `prioritized_variants` section of the JSON output. The complete JSON file additionally contains all transcript-level annotations returned by Variant Validator that are compatible with the queried variant (`c.220C>T`) and whose predicted protein consequence matches the protein-level description provided by the user (`p.Arg74Trp`).

Example case 2: Gene–transcript incoherence

A second example illustrates how the workflow detects inconsistencies between the information provided by the user and the annotation returned by the Variant Validator.

Considering the following input record extracted from the curated dataset:


```
{‘‘gene’’: ‘‘BRCA1’’,
 ‘‘rs’’: ‘‘’’,
 ‘‘hgvs_c’’: ‘‘c.470T>C’’,
 ‘‘hgvs_p’’: ‘‘’’,
 ‘‘hgvs_m’’: ‘‘’’
 ‘‘genomic_coord’’: ‘‘’’,
 ‘‘transcript’’: ‘‘NM_007194.4’’}
```

```
"prioritized_variants": {
  "input": "CFTR:c.220C>T",
  "transcript": "NM_000492.4 (protein_coding) MANE:Select",
  "gene": "CFTR",
  "strand": "+",
  "coordinates(gDNA/cDNA/protein)": "chr7:g.117509089C>T|c.220C>T|p.R74W",
  "region": "inside_[cds_in_exon_3]",
  "info": {
    "CSQN": "Missense",
    "dbsnp": "rs115545701(chr7:117509089C>T)",
    "reference_codon": "CGG",
    "alternative_codon": "TGG",
    "dbxref": "GeneID:1080,HGNC:1884,MIM:602421",
    "aliases": "NP_000483",
    "source": "RefSeq"
  },
  "hgvs_c_nucleo": "c.220C>T"
},
"alternative_transcripts": [],
"feedback": [],
"source_variant": {
  "gene": "CFTR",
  "rs": "",
  "hgvs_c": "c.220C>T",
  "hgvs_p": "p.Arg74Trp",
  "hgvs_m": "",
  "genomic_coord": "",
  "transcript": ""
},
"status": 3,
"coherence_checks": {
  "gene": true,
  "transcript": true,
  "hgvs_c": true,
  "hgvs_p": true
}
```


Figure 5.3: **JSON output generated after input reconstruction and prioritization.** The input variant is provided using the gene symbol together with both coding and protein HGVS descriptions. The workflow reconstructs the query `CFTR:c.220C>T`, performs validation across transcript databases, and returns a structured output containing the prioritized transcript, genomic coordinates, predicted consequence, codon change, and external database identifiers.

In this case the user provides both a gene symbol and a transcript identifier together with a coding HGVS description. According to the prioritization strategy described above, transcript-level information is considered, since it is more specific than the gene symbol. Therefore, the reconstructed query becomes:

`NM_007194.4:c.470T>C`

The validation is then performed using the transcript provided by the user. However, the transcript `NM_007194.4` corresponds to the gene *CHEK2*, while the user specified the gene *BRCA1*. As a result, a gene-level inconsistency is detected.

During the coherence validation stage, the gene inferred from the Variant Validator annotation is compared with the gene provided by the user. Since the two values do not match, the `gene` field in the `coherence_checks` structure is flagged accordingly. At the same time, the other fields (`transcript` and `hgvs_c`) remain coherent with the reconstructed input.

The system reports this inconsistency through the feedback field while preserving the annotation obtained from the Variant Validator. An example of the resulting structured output is shown in Figure 5.4.

5.2.2 Comparison of validation completeness between VEP REST API and the Variant Validator

Using the input prioritization strategy described in the previous section, the 2001 manually curated variants were reconstructed and sequentially submitted to the Variant Validator. Each variant was processed independently and the corresponding annotation was stored in a structured JSON output file containing the reconstructed input, prioritized variant representation, alternative transcripts, and coherence checks.

The execution was performed sequentially, with each variant analyzed through the set of attempts described in the prioritization strategy in Par. 5.2.1. The average execution time required to process a single variant was 0.369 seconds. As a consequence, the complete annotation of all 2001 variants required approximately 15 minutes. These results highlight the efficiency of the local validation workflow,

```
"prioritized_variants": {
  "input": "NM_007194.4:c.470T>C",
  "transcript": "NM_007194.4 (protein_coding) MANE:Select",
  "gene": "CHEK2",
  "strand": "-",
  "coordinates(gDNA/cDNA/protein)": "chr22:g.28725099A>G|c.470T>C|p.I157T",
  "region": "inside_[cds_in_exon_4]",
  "info": {
    "CSQN": "Missense",
    "dbsnp": "rs17879961(chr22:28725099A>G)",
    "reference_codon": "ATT",
    "alternative_codon": "ACT",
    "dbxref": "GeneID:11200,HGNC:16627,MIM:604373",
    "aliases": "NP_009125",
    "source": "RefSeq"
  },
  "hgvs_c_nucleo": "c.470T>C"
},
"alternative_transcripts": [],
"feedback": [
  "Incoherent gene: BRCA1 do not match with any genes found (CHEK2)"
],
"source_variant": {
  "gene": "BRCA1",
  "rs": "",
  "hgvs_c": "c.470T>C",
  "hgvs_p": "",
  "hgvs_m": "",
  "genomic_coord": "",
  "transcript": "NM_007194.4"
},
"status": 4,
"coherence_checks": {
  "gene": false,
  "transcript": true,
  "hgvs_c": true,
  "hgvs_p": true
}
```


Figure 5.4: **Example of gene-level incoherence detected during validation.** The user provides the gene *BRCA1* together with the transcript `NM_007194.4` and the coding HGVS description `c.470T>C`. The transcript corresponds to the gene *CHEK2*, generating a mismatch between the gene specified by the user and the gene inferred from the annotation. The inconsistency is reported through the `coherence_checks` structure and the feedback message while preserving the annotated variant information.

showing that large batches of variants can be processed within a short time while maintaining a fully automated pipeline.

A more detailed view of the execution time distribution is provided in Fig. 5.5, which reports the boxplots of the processing times computed over the 1956 variants that produced a valid output. Two representations are shown: the first includes all observations, providing a complete overview of the variability in execution times, while the second excludes a total of 229 outliers to improve readability of the central distribution.

It is important to note that execution time is not uniform across variants, but depends on the amount and type of input information provided. In particular, the number of transcripts associated with a given variant plays a key role in determining computational cost. Variants linked to genes with a high number of transcripts require a larger number of projections and consistency checks, resulting in longer processing times. Consequently, execution time can be considered approximately proportional to the number of transcripts evaluated during the validation process.

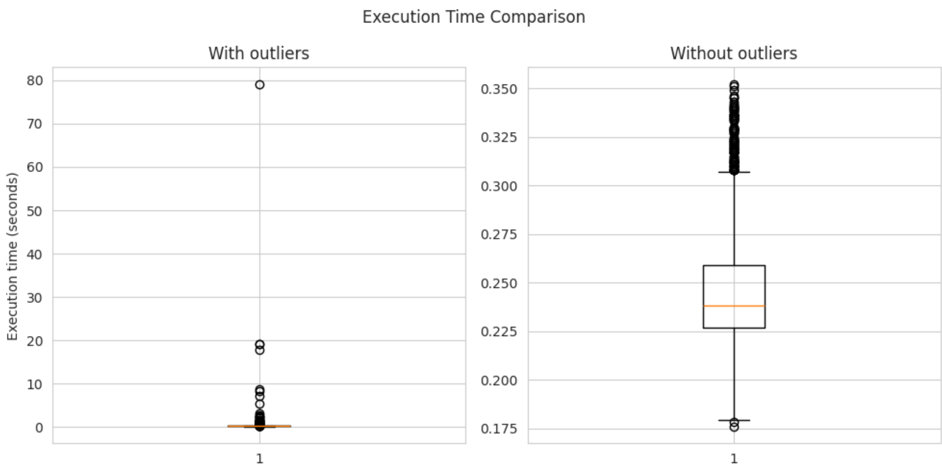

Figure 5.5: **Distribution of execution times for variant validation.** Boxplots of execution times computed over the 1956 variants that produced a valid output. The left panel includes all observations, providing a complete overview of variability, while the right panel shows the distribution after removal of outliers (IQR-based filtering), highlighting the central tendency of execution times. The presence of outliers reflects cases with increased computational complexity, associated with variants linked to genes with a higher number of transcripts.

Once the Variant Validator annotations were generated, the resulting outputs were compared with the annotations obtained through the VEP REST API calls.

For each variant, the outputs produced by the two tools were compared field by field. The comparison focused on the following annotation fields: gene, transcript, hgvs_c, hgvs_p, genomic_coord, and rs.

For each field, the mismatches were classified into three possible categories:

- only VEP REST API: the field was completed only by the VEP REST API, while the Variant Validator did not provide a corresponding value;
- only Variant Validator: the field was completed only by the Variant Validator, while the VEP REST API did not return any value;
- different values: both tools returned a value for the field, but the annotations differed between the two tools.

This comparison allows the evaluation of how the two validation strategies differ in terms of information retrieval and automatic completion of missing fields.

The distribution of mismatches across the analyzed fields is shown in Figure 5.6. The heatmap reports the percentage distribution of mismatch types for each annotation field.

Importantly, these percentages are not computed over the total number of analyzed variants (1956), but only over the subset of variants showing a mismatch for the corresponding field. Therefore, for each field, the three percentages shown in the heatmap describe how the discordant cases are distributed among the categories only VEP REST API, only Variant Validator, and different values.

To contextualize these distributions, Table 5.2 reports the overall agreement between the two tools for each annotation field across the full set of 1956 analyzed variants.

The results show that the two tools are overall highly consistent across most annotation fields, as highlighted by the high match rates reported in Table 5.2. In particular, agreement exceeds 95% for the gene, hgvs_c, and hgvs_p fields, indicating that both tools generally converge toward the same variant representation when sufficient input information is available. Lower agreement is observed for the transcript (85.28%) and rs (83.33%) fields, while the genomic_coord field shows a

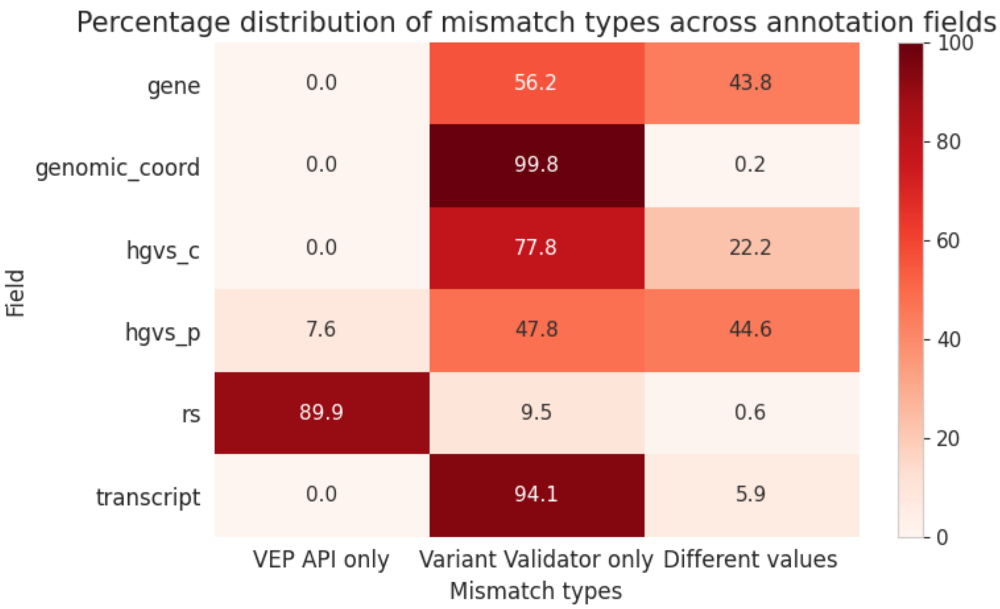

Figure 5.6: **Distribution of annotation mismatches between VEP API and the Variant Validator.** The heatmap shows the percentage distribution of mismatch types observed for each annotation field when comparing the outputs produced by the VEP REST API and the Variant Validator. Percentages are computed within the subset of variants showing a mismatch for the corresponding field, and therefore do not refer to the total number of analyzed variants.

Table 5.2: Agreement between the VEP REST API and the Variant Validator across the 1956 analyzed variants, reported separately for each annotation field.

Field	**Match rate**	**Matches**	**Total**	**Accuracy (%)**
gene	0.975460	1908	1956	97.546
hgvs_c	0.958589	1875	1956	95.859
hgvs_p	0.952965	1864	1956	95.297
transcript	0.852761	1668	1956	85.276
rs	0.833333	1630	1956	83.333
genomic_coord	0.033231	65	1956	3.323

very low match rate (3.32%), reflecting a design difference rather than a true inconsistency.

Despite this overall agreement, the analysis of mismatches provides important insights into the different behaviors of the two validation strategies.

For the gene field, although the agreement is very high (97.55%), the observed mismatches are mainly due to two factors. First, the VEP REST API often does not populate the gene symbol unless it is explicitly provided, whereas the Variant Validator systematically reports the gene associated with the mapped transcript or genomic location, leading to cases classified as only Variant Validator. Second, mismatches categorized as different values are primarily attributable to inconsistencies in the user-provided input, where the gene symbol and transcript identifier do not correspond to the same gene. In these cases, the Variant Validator resolves the ambiguity by relying on the transcript, which uniquely defines the gene, thereby revealing the inconsistency. This behavior is consistent with the coherence checks described in Par. 5.3 and with the example illustrated in Figure 5.3.

Regarding the transcript field, the lower agreement (85.28%) reflects a systematic difference between the two tools. The VEP REST API often does not populate transcript identifiers unless they are explicitly provided in the input query, whereas the Variant Validator systematically reports all compatible transcript-level annotations. As a consequence, most mismatches correspond to cases where the transcript field is completed only by the Variant Validator. Cases classified as different values are consistently attributable to inconsistencies in the user input, where the gene and transcript specified by the user are not compatible.

For the hgvs_c field, the agreement remains high (95.86%), confirming that both tools generally converge on the same coding-level representation. However, mismatches arise mainly from two sources. On one hand, the VEP REST API often does not return a coding HGVS description unless it is explicitly provided, resulting in cases completed only by the Variant Validator. On the other hand, cases classified as different values are mostly attributable to HGVS normalization issues in the original input. For example, the variant `AASS:c.747_748AG>CT` is not HGVS-compliant and is automatically normalized by the Variant Validator to the correct representation `c.747_748delinsCT`, leading to discrepancies with the non-normalized representation returned by the VEP API.

A similar behavior is observed for the hgvs_p field, where agreement remains high (95.30%). As for the coding-level representation, the VEP REST API often does not populate protein-level HGVS descriptions unless they are explicitly provided or directly derivable, whereas the Variant Validator systematically reconstructs them. In addition, mismatches frequently arise from differences in the representation of frameshift variants. The Variant Validator follows HGVS recommendations by explicitly reporting the length of the altered reading frame and the position of the newly introduced stop codon, whereas the VEP REST API often provides truncated annotations limited to the *fs* suffix. This difference in representation leads to a substantial fraction of different values despite the underlying biological effect being consistent.

For the genomic_coord field, the very low agreement (3.32%) is not indicative of a limitation of the Variant Validator, but rather reflects a deliberate design choice in the VarChat autocompletion workflow. The VEP-based autocompletion procedure does not explicitly populate genomic coordinates unless they are already present in the original trimmed input. This design choice reflects the fact that genomic coordinates are rarely reported in the scientific literature. Instead, variants are typically described using gene symbols, transcript identifiers, HGVS coding or protein notations (c./p.), or rsIDs.

Finally, for the rs field, the lower agreement (83.33%) is mainly due to differences in the underlying dbSNP resources. A substantially larger amount of rsID information is retrieved through the VEP REST API, as the internal dbSNP VCF files used by the Variant Validator are not the most recent releases. This limitation is mainly related to local storage constraints and could be easily overcome by deploying the validation workflow on scalable cloud infrastructures (e.g., AWS), where the latest dbSNP datasets can be integrated without storage limitations.

In conclusion, the comparative analysis demonstrates that the two tools achieve a high level of concordance across the majority of annotation fields, with discrepancies that are largely explainable by design choices, input inconsistencies, or differences in normalization strategies rather than true annotation errors. Importantly, the Variant Validator consistently exhibits a more deterministic and completeness-oriented behavior, systematically enriching missing information and enforcing HGVS-compliant representations.

From an integration perspective, the three initial objectives have been successfully achieved. The Variant Validator performs at least comparably, and in several aspects more robustly, than the current system; it operates within acceptable time constraints; and it removes the dependency on external services, thereby improving reliability and reproducibility. Altogether, these results confirm that the Variant Validator represents a suitable and effective solution for integration within the VarChat autocompletion workflow.

Chapter 6

Conclusions and future directions

The objective of this work was to identify and develop a variant validation framework independent from external services, capable of handling heterogeneous input formats (HGVS, VCF-like coordinates, and rsIDs), and able to minimize ambiguity in variant representation across genomic, transcript, and protein levels.

To this end, several widely used tools were initially evaluated, including LOVD Syntax Checker, Mutalyzer, VariantValidator, and VEP. This preliminary analysis highlighted a set of common limitations. Tools such as Mutalyzer and VEP rely on external services or APIs, reducing offline usability. On the other hand, tools like VariantValidator and SnpEff do not provide a complete and consistent cross-mapping between genomic (g.), coding (c.), and protein (p.) representations. Additionally, ambiguity in transcript selection further limit their applicability in a fully automated validation pipeline.

Based on these observations, the focus converged on TransVar as the most suitable candidate for the aim of the work. TransVar provides a solid foundation for cross-level variant projection and operates locally, making it compatible with the requirement of independence from external resources. Its implementation in Python, with publicly available and modifiable source code, makes it particularly suitable for extension and integration within custom validation workflows. TransVar original implementation presented several limitations, particularly in the handling of untranslated regions, transcript diversity, and non-coding genes.

To address these issues, the internal preprocessing and validation logic of

TransVar were substantially extended. Support for non-coding transcripts and genes was introduced, and transcript models were enriched to explicitly represent segmented UTR regions. These modifications transformed the original CDS-centric approach into a full structure-aware representation of transcripts, enabling accurate and HGVS-compliant annotation across all variant types. Finally, improvements were made to transcript handling, including prioritization for MANE annotations. The resulting extended framework has been referred to as the Variant Validator

The impact of these changes was systematically evaluated through benchmarking analyses. Comparisons between TransVar and the Variant Validator against external tools such as SnpEff, revealed clear improvements. Differences were analyzed and categorized across multiple dimensions, including the number of shared transcripts, discrepancies in variant effect classification, and inconsistencies in c. and p. HGVS representations. Following the improvements, a reduction in mismatch categories was observed, particularly for variants located in UTR regions, which were previously misclassified due to the lack of transcript structure awareness.

Finally, the Variant Validator will be integrated into a real-world validation workflow within VarChat: the first generative AI–based platform developed by enGenome srl that enables users to retrieve and summarize the scientific literature associated with a specific genetic variant. In this context, the contribution is not limited to simple input reconstruction, but extended to the design of a structured nomenclature validation and prioritization pipeline.

First, a flexible input reconstruction strategy was defined to handle incomplete or heterogeneous variant descriptions. User-provided inputs may include partial information (e.g., only gene and protein change, or only genomic coordinates), and therefore require normalization and completion. Available fields are combined to obtain a consistent internal representation of the variant.

Beyond normalization and completion, a prioritization logic has been introduced to ensure that the most specific and informative representation is always returned to the user. Different input fields (gene, transcript, genomic coordinates, rsID, HGVS notations) are evaluated according to their level of specificity and reliability, and the output is structured to privilege the highest-resolution information available. This prevents ambiguous or redundant representations and guarantees consistency across genomic (g.), coding (c.), and protein (p.) levels.

To further improve robustness, a system of validation flags was implemented for gene, transcript, and HGVS fields. These flags are used to detect and correct inconsistencies between user-provided inputs and the annotations generated by the Variant Validator. For instance, mismatches in transcript identifiers, gene names, or HGVS expressions can be identified and resolved, ensuring that the final output reflects a biologically and syntactically coherent variant representation.

The effectiveness of this workflow was evaluated on the VarChat benchmark dataset. A direct comparison between the Variant Validator-based pipeline and the current VEP REST API-based validation and autocompletion strategy showed that the proposed approach achieves a higher level of field completion. At the same time, it maintains full consistency with VEP in terms of variant validation, producing equivalent results while avoiding dependence on external services.

This demonstrates that the integration of an improved, fully local variant validation framework can both enhance completeness and preserve reliability, representing a strong alternative to API-dependent solutions in real-world applications.

Overall, this work demonstrates that achieving reliable variant validation requires not only accurate coordinate transformations, but also a comprehensive and biologically consistent modeling of transcript structure, combined with a robust strategy for handling real-world, partial or ambiguous inputs.

Despite the improvements introduced in this work, some limitations remain and highlight possible directions for future development.

The primary limitation concerns the nature of the validation process itself. While the actual Variant Validator framework is able to validate and normalize variant representations across genomic, transcript, and protein levels, it does not provide explicit feedback on the origin of validation failures. In cases where the input variant is incorrect or inconsistent-such as when the reference base does not match the selected genome assembly or when coordinates are misaligned-the system typically fails to validate the variant without offering actionable suggestions to the user.

This lack of diagnostic capability limits usability, especially in prone-to-errors real-world scenarios. A possible extension of this work would therefore involve the development of an error interpretation layer capable of identifying the most likely reason for validation failure and suggesting possible corrections. For example, the system could detect mismatches between reference alleles and genome assemblies, suggest the correct reference sequence when the wrong genome build is being used, or it could recommend compatible transcripts when the one provided by the user is not valid.

A second limitation concerns the representation of variant effects. Although the actual Variant Validator workflow ensures accurate coordinate projection and HGVS-compliant annotation, its internal ontology for variant effect classification remains relatively limited compared to widely used tools such as SnpEff and VEP. These tools provide a richer and more standardized set of consequence terms, enabling finer-grained interpretation of variant impact.

Future work could address this gap by integrating a more comprehensive effect ontology, either by extending the internal classification system or by mapping the Variant Validator outputs to established ontologies such as Sequence Ontology terms [42]. This would enhance interoperability with existing genomic pipelines and improve the interpretability of results.

The last limitation concerns the scope of supported variant types. At present, the Variant Validator framework is primarily designed to handle small-scale variants: single nucleotide variants (SNVs) and short insertions and deletions (INDELs). While these represent the majority of variants typically analyzed in many workflows, more complex classes of genomic alterations are not currently supported.

In particular, Short Tandem Repeats (STRs), structural variants (SVs) and copy number variations (CNVs) fall outside the scope of the current implementation. These variant types are increasingly relevant in both research and clinical genomics, as they can have significant functional and phenotypic impact.

Future work should therefore focus on extending the framework to support the validation and representation of these more complex variants. This would mainly require adapting coordinate mapping strategies in order to describe large-scale genomic rearrangements.

Bibliography

[1] S. R. Banoon, T. S. Salih, and A. Ghasemian, "Genetic mutations and major human disorders: A review," *Egyptian Journal of Chemistry*, vol. 65, no. 2, pp. 571–589, 2022.

[2] . G. P. Consortium *et al.*, "A global reference for human genetic variation," *Nature*, vol. 526, no. 7571, p. 68, 2015.

[3] Y. Zhang and Z.-Y. Wu, "Gene therapy for monogenic disorders: challenges, strategies, and perspectives," *Journal of Genetics and Genomics*, vol. 51, no. 2, pp. 133–143, 2024.

[4] J. X. Chong, K. J. Buckingham, S. N. Jhangiani, C. Boehm, N. Sobreira, J. D. Smith, T. M. Harrell, M. J. McMillin, W. Wiszniewski, T. Gambin, *et al.*, "The genetic basis of mendelian phenotypes: discoveries, challenges, and opportunities," *The American Journal of Human Genetics*, vol. 97, no. 2, pp. 199–215, 2015.

[5] The Gene Home, "Examples of genetic diseases." `https://www.thegenehome.com/basics-of-genetics/disease-examples`.

[6] A. M. Kanzi, J. E. San, B. Chimukangara, E. Wilkinson, M. Fish, V. Ramsuran, and T. De Oliveira, "Next generation sequencing and bioinformatics analysis of family genetic inheritance," *Frontiers in genetics*, vol. 11, p. 544162, 2020.

[7] K. Lohmann and C. Klein, "Next generation sequencing and the future of genetic diagnosis," *Neurotherapeutics*, vol. 11, no. 4, pp. 699–707, 2014.

[8] M. L. Metzker, "Sequencing technologies — the next generation," *Nature Reviews Genetics*, vol. 11, pp. 31–46, 2010.

[9] S. Sawyer, T. Hartley, D. Dyment, C. Beaulieu, J. Schwartzentruber, A. Smith, H. Bedford, G. Bernard, F. Bernier, B. Brais, *et al.*, "Utility of whole-exome sequencing for those near the end of the diagnostic odyssey: time to address gaps in care," *Clinical genetics*, vol. 89, no. 3, pp. 275–284, 2016.

[10] B. Sikkema-Raddatz, L. F. Johansson, E. N. de Boer, R. Almomani, L. G. Boven, M. P. van den Berg, K. Y. van Spaendonck-Zwarts, J. P. van Tintelen, R. H. Sijmons, J. D. Jongbloed, *et al.*, "Targeted next-generation sequencing can replace sanger sequencing in clinical diagnostics," *Human mutation*, vol. 34, no. 7, pp. 1035–1042, 2013.

[11] D. G. MacArthur, T. Manolio, D. Dimmock, H. Rehm, J. Shendure, G. Abecasis, D. Adams, R. Altman, S. Antonarakis, E. Ashley, *et al.*, "Guidelines for investigating causality of sequence variants in human disease," *Nature*, vol. 508, no. 7497, pp. 469–476, 2014.

[12] J. T. den Dunnen, "Describing sequence variants using hgvs nomenclature," in *Genotyping: Methods and Protocols*, pp. 243–251, Springer, 2016.

[13] R. R. Haraksingh and M. P. Snyder, "Impacts of variation in the human genome on gene regulation," *Journal of molecular biology*, vol. 425, no. 21, pp. 3970–3977, 2013.

[14] C. Børsting, V. Pereira, J. D. Andersen, and N. Morling, "Single nucleotide polymorphism," *A Guide to Forensic DNA Profiling*, p. 205, 2016.

[15] Q. Wang, E. Pierce-Hoffman, B. B. Cummings, J. Alföldi, L. C. Francioli, L. D. Gauthier, A. J. Hill, A. H. O'Donnell-Luria, *et al.*, "Landscape of multi-nucleotide variants in 125,748 human exomes and 15,708 genomes," *Nature communications*, vol. 11, no. 1, p. 2539, 2020.

[16] J. M. Mullaney, R. E. Mills, W. S. Pittard, and S. E. Devine, "Small insertions and deletions (indels) in human genomes," *Human molecular genetics*, vol. 19, no. R2, pp. R131–R136, 2010.

[17] L. Feuk, A. R. Carson, and S. W. Scherer, "Structural variation in the human genome," *Nature Reviews Genetics*, vol. 7, no. 2, pp. 85–97, 2006.

[18] M. Zarrei, J. R. MacDonald, D. Merico, and S. W. Scherer, "A copy number variation map of the human genome," *Nature reviews genetics*, vol. 16, no. 3, pp. 172–183, 2015.

[19] A. V. Nesta, D. Tafur, and C. R. Beck, "Hotspots of human mutation," *Trends in Genetics*, vol. 37, no. 8, pp. 717–729, 2021.

[20] G. R. Brown, V. Hem, K. S. Katz, M. Ovetsky, C. Wallin, O. Ermolaeva, I. Tolstoy, T. Tatusova, K. D. Pruitt, D. R. Maglott, *et al.*, "Gene: a gene-centered information resource at ncbi," *Nucleic acids research*, vol. 43, no. D1, pp. D36–D42, 2015.

[21] N. A. O'Leary, M. W. Wright, J. R. Brister, S. Ciufo, D. Haddad, R. McVeigh, B. Rajput, B. Robbertse, B. Smith-White, D. Ako-Adjei, *et al.*, "Reference sequence (refseq) database at ncbi: current status, taxonomic expansion, and functional annotation," *Nucleic acids research*, vol. 44, no. D1, pp. D733–D745, 2016.

[22] T. Hubbard, D. Barker, E. Birney, G. Cameron, Y. Chen, L. Clark, T. Cox, J. Cuff, V. Curwen, T. Down, *et al.*, "The ensembl genome database project," *Nucleic acids research*, vol. 30, no. 1, pp. 38–41, 2002.

[23] J. Navarro Gonzalez, A. S. Zweig, M. L. Speir, D. Schmelter, K. R. Rosenbloom, B. J. Raney, C. C. Powell, L. R. Nassar, N. D. Maulding, C. M. Lee, *et al.*, "The ucsc genome browser database: 2021 update," *Nucleic acids research*, vol. 49, no. D1, pp. D1046–D1057, 2021.

[24] V. Schneider and D. Church, "Genome reference consortium," *The NCBI Handbook [Internet], 2nd edn. National Center for Biotechnology Information (US), Bethesda, MD*, 2013.

[25] P. A. Kitts, D. M. Church, F. Thibaud-Nissen, J. Choi, V. Hem, V. Sapojnikov, R. G. Smith, T. Tatusova, C. Xiang, A. Zherikov, *et al.*, "Assembly: a resource for assembled genomes at ncbi," *Nucleic acids research*, vol. 44, no. D1, pp. D73–D80, 2016.

[26] V. S. Aggarwal, S. Guha, M. S. Lebo, K. Retterer, C. Scharfe, L. J. Bean, A. L. Q. A. Committee, *et al.*, "Laboratory considerations for grch37 to grch38 reference genome transition: A laboratory quality assurance bulletin of the american college of medical genetics and genomics (acmg)," *Genetics in Medicine Open*, vol. 3, p. 103433, 2025.

[27] R. K. Hart, I. F. Fokkema, M. DiStefano, R. Hastings, J. F. Laros, R. Taylor, A. H. Wagner, and J. T. den Dunnen, "Hgvs nomenclature 2024: improvements to community engagement, usability, and computability," *Genome medicine*, vol. 16, no. 1, p. 149, 2024.

[28] R. Dalgleish, P. Flicek, F. Cunningham, A. Astashyn, R. E. Tully, G. Proctor, Y. Chen, W. M. McLaren, P. Larsson, B. W. Vaughan, *et al.*, "Locus reference genomic sequences: an improved basis for describing human dna variants," *Genome medicine*, vol. 2, no. 4, p. 24, 2010.

[29] L. Y. Geer, A. Marchler-Bauer, R. C. Geer, L. Han, J. He, S. He, C. Liu, W. Shi, and S. H. Bryant, "The ncbi biosystems database," *Nucleic acids research*, vol. 38, no. suppl_1, pp. D492–D496, 2010.

[30] A. T. Kraja, C. Liu, J. L. Fetterman, M. Graff, C. T. Have, C. Gu, L. R. Yanek, M. F. Feitosa, D. E. Arking, D. I. Chasman, *et al.*, "Associations of mitochondrial and nuclear mitochondrial variants and genes with seven metabolic traits," *The American Journal of Human Genetics*, vol. 104, no. 1, pp. 112–138, 2019.

[31] J. S. Mattick and I. V. Makunin, "Non-coding rna," *Human molecular genetics*, vol. 15, no. suppl_1, pp. R17–R29, 2006.

[32] J. T. Den Dunnen, R. Dalgleish, D. R. Maglott, R. K. Hart, M. S. Greenblatt, J. McGowan-Jordan, A.-F. Roux, T. Smith, S. E. Antonarakis, P. E. Taschner,

et al., "Hgvs recommendations for the description of sequence variants: 2016 update," *Human mutation*, vol. 37, no. 6, pp. 564–569, 2016.

[33] I. F. Fokkema, P. E. Taschner, G. C. Schaafsma, J. Celli, J. F. Laros, and J. T. den Dunnen, "Lovd v. 2.0: the next generation in gene variant databases," *Human mutation*, vol. 32, no. 5, pp. 557–563, 2011.

[34] M. Lefter, J. K. Vis, M. Vermaat, J. T. den Dunnen, P. E. Taschner, and J. F. Laros, "Mutalyzer 2: next generation hgvs nomenclature checker," *Bioinformatics*, vol. 37, no. 18, pp. 2811–2817, 2021.

[35] P. J. Freeman, R. K. Hart, L. J. Gretton, A. J. Brookes, and R. Dalgleish, "Variantvalidator: Accurate validation, mapping, and formatting of sequence variation descriptions," *Human mutation*, vol. 39, no. 1, pp. 61–68, 2018.

[36] M. Wang, K. M. Callenberg, R. Dalgleish, A. Fedtsov, N. K. Fox, P. J. Freeman, K. B. Jacobs, P. Kaleta, A. J. McMurry, A. Prlić, *et al.*, "hgvs: A python package for manipulating sequence variants using hgvs nomenclature: 2018 update," *Human mutation*, vol. 39, no. 12, pp. 1803–1813, 2018.

[37] P. Cingolani, A. Platts, L. L. Wang, M. Coon, T. Nguyen, L. Wang, S. J. Land, X. Lu, and D. M. Ruden, "A program for annotating and predicting the effects of single nucleotide polymorphisms, snpeff: Snps in the genome of drosophila melanogaster strain w1118; iso-2; iso-3," *fly*, vol. 6, no. 2, pp. 80–92, 2012.

[38] W. Zhou, T. Chen, Z. Chong, M. A. Rohrdanz, J. M. Melott, C. Wakefield, J. Zeng, J. N. Weinstein, F. Meric-Bernstam, G. B. Mills, *et al.*, "Transvar: a multilevel variant annotator for precision genomics," *Nature methods*, vol. 12, no. 11, pp. 1002–1003, 2015.

[39] A. Frankish, M. Diekhans, I. Jungreis, J. Lagarde, J. E. Loveland, J. M. Mudge, C. Sisu, J. C. Wright, J. Armstrong, I. Barnes, *et al.*, "Gencode 2021," *Nucleic acids research*, vol. 49, no. D1, pp. D916–D923, 2021.

[40] Wanding Zhou, "Transvar documentation." `https://transvar.readthedocs.io/en/latest/`.

[41] S. E. Hunt, B. Moore, R. M. Amode, I. M. Armean, D. Lemos, A. Mushtaq, A. Parton, H. Schuilenburg, M. Szpak, A. Thormann, *et al.*, “Annotating and prioritizing genomic variants using the ensembl variant effect predictor—a tutorial,” *Human mutation*, vol. 43, no. 8, pp. 986–997, 2022.

[42] K. Eilbeck, S. E. Lewis, C. J. Mungall, M. Yandell, L. Stein, R. Durbin, and M. Ashburner, “The sequence ontology: a tool for the unification of genome annotations,” *Genome biology*, vol. 6, no. 5, p. R44, 2005.

[43] S. Gudmundsson, M. Singer-Berk, N. A. Watts, W. Phu, J. K. Goodrich, M. Solomonson, G. A. D. Consortium, H. L. Rehm, D. G. MacArthur, and A. O’Donnell-Luria, “Variant interpretation using population databases: Lessons from gnomad,” *Human mutation*, vol. 43, no. 8, pp. 1012–1030, 2022.

[44] L. Clarke, X. Zheng-Bradley, R. Smith, E. Kulesha, C. Xiao, I. Toneva, B. Vaughan, D. Preuss, R. Leinonen, M. Shumway, *et al.*, “The 1000 genomes project: data management and community access,” *Nature methods*, vol. 9, no. 5, pp. 459–462, 2012.

[45] M. J. Landrum, J. M. Lee, G. R. Riley, W. Jang, W. S. Rubinstein, D. M. Church, and D. R. Maglott, “Clinvar: public archive of relationships among sequence variation and human phenotype,” *Nucleic acids research*, vol. 42, no. D1, pp. D980–D985, 2014.

[46] J. G. Tate, S. Bamford, H. C. Jubb, Z. Sondka, D. M. Beare, N. Bindal, H. Boutselakis, C. G. Cole, C. Creatore, E. Dawson, *et al.*, “Cosmic: the catalogue of somatic mutations in cancer,” *Nucleic acids research*, vol. 47, no. D1, pp. D941–D947, 2019.

[47] M. Blum, H.-Y. Chang, S. Chuguransky, T. Grego, S. Kandasaamy, A. Mitchell, G. Nuka, T. Paysan-Lafosse, M. Qureshi, S. Raj, *et al.*, “The interpro protein families and domains database: 20 years on,” *Nucleic acids research*, vol. 49, no. D1, pp. D344–D354, 2021.

[48] J. Wu, Z. Cui, V. S. Sheng, P. Zhao, D. Su, and S. Gong, “A comparative study of sift and its variants,” *Measurement science review*, vol. 13, no. 3, pp. 122–131, 2013.

[49] I. Adzhubei, D. M. Jordan, and S. R. Sunyaev, "Predicting functional effect of human missense mutations using polyphen-2," *Current protocols in human genetics*, vol. 76, no. 1, pp. 7–20, 2013.

[50] M. Schubach, T. Maass, L. Nazaretyan, S. Röner, and M. Kircher, "Cadd v1. 7: using protein language models, regulatory cnns and other nucleotide-level scores to improve genome-wide variant predictions," *Nucleic acids research*, vol. 52, no. D1, pp. D1143–D1154, 2024.

[51] J.-M. de Sainte Agathe, M. Filser, B. Isidor, T. Besnard, P. Gueguen, A. Perrin, C. Van Goethem, C. Verebi, M. Masingue, J. Rendu, *et al.*, "Spliceai-visual: a free online tool to improve spliceai splicing variant interpretation," *Human Genomics*, vol. 17, no. 1, p. 7, 2023.

[52] D. Merkel *et al.*, "Docker: lightweight linux containers for consistent development and deployment," *Linux j*, vol. 239, no. 2, p. 2, 2014.

[53] C. Pintoiu, "Mongodb vs documentdb," *BigStep, date found via Google as Apr*, vol. 1, 2019.

[54] C. F. Wright, D. R. FitzPatrick, J. S. Ware, H. L. Rehm, and H. V. Firth, "Importance of adopting standardized mane transcripts in clinical reporting," *Genetics in Medicine*, vol. 25, no. 2, 2023.

[55] G. Nicora, S. Zucca, I. Limongelli, R. Bellazzi, and P. Magni, "A machine learning approach based on acmg/amp guidelines for genomic variant classification and prioritization," *Scientific reports*, vol. 12, no. 1, p. 2517, 2022.

[56] M. J. Landrum, S. Chitipiralla, G. R. Brown, C. Chen, B. Gu, J. Hart, D. Hoffman, W. Jang, K. Kaur, C. Liu, *et al.*, "Clinvar: improvements to accessing data," *Nucleic acids research*, vol. 48, no. D1, pp. D835–D844, 2020.

[57] F. De Paoli, S. Berardelli, I. Limongelli, E. Rizzo, and S. Zucca, "Varchat: the generative ai assistant for the interpretation of human genomic variations," *Bioinformatics*, vol. 40, no. 4, p. btae183, 2024.

[58] F. De Paoli, S. Berardelli, A. Tudisco, A. Blindu, E. Parimbelli, and S. Zucca, "Generative ai meets genomics: Varchat, a rag-based approach for literature-driven variant summarization," in *International Conference on Artificial Intelligence in Medicine*, pp. 127–131, Springer, 2025.

[59] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *IEEE transactions on knowledge and data engineering*, vol. 34, no. 1, pp. 50–70, 2020.

[60] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, H. Wang, H. Wang, *et al.*, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, vol. 2, no. 1, p. 32, 2023.

Ringraziamenti

Arrivato al termine di questi cinque anni intensi e significativi, mi viene spontaneo fermarmi un momento a riflettere sul percorso che mi ha portato fin qui oggi. Sembra ieri quando ho iniziato questo cammino, pieno di dubbi e incertezze sulla scelta fatta e su ciò che mi avrebbe riservato il futuro. E invece, quasi senza accorgermene, il tempo è passato e mi ha condotto fino a questo giorno.

In questi anni ho vissuto tante esperienze che mi hanno fatto crescere, non solo dal punto di vista accademico ma anche personale. Ho avuto la fortuna di incontrare nuove persone, conoscere volti che hanno lasciato un segno profondo in questo percorso contribuendo a rendermi la persona che sono oggi. Allo stesso tempo, è stato fondamentale aver mantenuto salde e preziose le amicizie di sempre, che mi hanno accompagnato lungo tutto il cammino. L'affetto e il sostegno ricevuti, nei momenti di entusiasmo come in quelli di difficoltà, sono stati il vero motore che mi ha spinto ad andare avanti e a non perdere mai di vista l'obiettivo.

Per questo motivo, desidero dedicare queste ultime righe a tutte le persone che, in modi diversi, hanno contribuito a rendere possibile questo importante traguardo raggiunto.

In primis, desidero ringraziare sinceramente il mio relatore Prof. Paolo Magni per avermi guidato lungo tutti questi mesi sempre con costanza e professionalità offrendo preziosi suggerimenti e indicazioni per lo sviluppo e il miglioramento di questo lavoro. Nel corso di questi anni di studio, i suoi insegnamenti mi hanno aiutato a sviluppare un modo di pensare più critico e consapevole che mi ha accompagnato durante tutto il percorso universitario e che continuerà ad essere un' importante linea guida anche per il futuro. Ringrazio anche il Prof. Lorenzo Pasotti per avermi introdotto e fatto appassionare al mondo della bioinformatica.

Ringrazio profondamente la mia correlatrice Ing. Susanna Zucca per avermi

permesso di intraprendere questa esperienza di internato di tesi e per avermi sempre seguito con costanza e dedizione in tutte le fasi di progettazione e sviluppo del lavoro. Ringrazio anche l' Ing. Ettore Rizzo, l'Ing. Ivan Limongelli e tutti i membri di enGenome per la calorosa accoglienza in azienda. L'ambiente stimolante e la disponibilità delle persone con cui ho avuto il piacere di lavorare hanno reso questa esperienza particolarmente formativa, permettendomi di crescere sia dal punto di vista professionale che personale.

Un sentito ringraziamento va alla mia correlatrice Silvia Berardelli che fin dall'inizio si è sempre impegnata fornendomi continuo aiuto e supporto. La sua precisione, dedizione e pazienza sono per me un punto di riferimento ed un modello da prendere come esempio. Sono stato molto fortunato a conoscerla: nel tempo ho scoperto in lei non solo una figura di riferimento professionale, ma anche una persona di grande disponibilità ed un'amica sincera con la quale ho potuto confidarmi liberamente. Nei momenti di difficoltà ha sempre creduto in me e questo è stato fondamentale. Un doveroso grazie va anche all' Ing. Federica De Paoli che, nel corso di questi mesi, ha seguito con attenzione e premura l'andamento di questo lavoro rendendosi sempre disponibile a offrire consigli e indicazioni preziose su come procedere al meglio.

Sono grato di aver conosciuto Andrei Blindu e Alessia Tudisco, che in questi mesi mi hanno costantemente sostenuto e spronato. Grazie per avermi accompagnato in questo percorso e per aver spesso alleggerito, con la vostra presenza, i momenti di tensione e di insicurezza. Sono molto grato di avervi incontrati e di potervi oggi chiamare amici.

Desidero spendere qualche parola per gli amici che ho avuto la fortuna di conoscere durante gli anni universitari e che mi sono sempre stati vicini, anche in questi ultimi due anni. Molti di questi rapporti sono nati durante gli anni della triennale e, nonostante nel corso della magistrale ci siamo trovati in città e percorsi diversi, il nostro legame è comunque rimasto forte. Le occasioni per ritrovarci e stare assieme non sono mai mancate e hanno sempre dimostrato che la distanza, in fondo, non è mai stata un problema. Tra tutti, desidero dedicare un pensiero particolare alle seguenti persone:

A Filippo, un amico col quale fin dall'inizio della triennale sono riuscito a legare. Il tempo passato assieme a raccontarci le nostre varie vicissitudini e interessi reciproci è stato proficuo e stimolante. Ti ringrazio per aver sempre creduto in me anche quando io stesso temevo di non farcela e per avermi infuso quella fiducia di cui spesso mancavo.

A Lau, da me scherzosamente definita “la donna da sposare”. Grazie per avermi sempre ascoltato nei momenti di bisogno e di esserti sempre voluta confidare con me. Le nostre videochiamate sono state momenti speciali che mi hanno fatto capire quanto siamo rimasti legati nonostante la distanza. Sapere di poterti avere accanto è stato un grande conforto nel corso di questi anni.

A Giuli, un’amica su cui ho sempre potuto contare e con la quale ho sempre condiviso tanti bei momenti. Grazie per tutte le volte in cui mi hai accolto a casa tua dopo lavoro per una pizza in compagnia: sono stati momenti semplici ma autentici, che hanno reso più leggeri e piacevoli anche i giorni più stancanti. La tua presenza e il tuo affetto nei miei confronti hanno saputo trasformare, quasi senza accorgermene, i momenti più difficili in momenti di serenità che ricorderò sempre con affetto.

A Michele, Anna, Bianca e Gabriele. Le feste organizzate in questi anni ma anche i semplici aperitivi di ritrovo che organizzavate quando tornavate per un breve periodo a Pavia, mi hanno sempre dimostrato l’affetto e il calore che ci lega come gruppo. Anche nei momenti più semplici, siamo sempre riusciti a ritrovarci e a condividere tempo e risate assieme.

A Gian e Giorgia, forse le due persone più capaci di strapparmi un sorriso in ogni momento. Con voi non sono mai mancati momenti di divertimento e di leggerezza, ma allo stesso tempo avete sempre saputo ascoltarmi e consigliarmi nei vari momenti lungo tutto questo percorso.

A Carola, Ludovico, Letizia e Federico. Vi ringrazio per l’amicizia nata e cresciuta nel corso di questi 5 anni. Con voi ho condiviso molte giornate tra lezioni, ore passate in aula studio e preparazioni agli esami, affrontando insieme ogni nuova sfida e difficoltà. Ma accanto ai momenti di studio non sono mai mancati quelli di svago e di leggerezza, che hanno reso questa amicizia ancora più speciale.

Un ringraziamento sincero va al “circolo” di amici vogheresi che conosco da letteralmente una vita: Pie, Monta, Giacomo, Giorgio, Sandro, Marco, Gogo, Bughi, Simo. Con molti di voi ci conosciamo dai tempi dell’asilo e ancora oggi siamo legati da un’amicizia indissolubile che mi ha portato ad allietare le giornate (e soprattutto le serate) di questi anni e a comprendere cosa sia la vera e genuina amicizia.

Una menzione doverosa e unica va al "Capo Supremo degli Emmi": Emma. Sei un'amica davvero speciale, sempre solare, altruista e sincera. Hai il raro dono di saper mettere gli altri al primo posto, di ascoltare e capire anche quando le parole non bastano. In ogni momento hai sempre saputo esserci, con un consiglio, una parola gentile o semplicemente con la tua presenza. La tua gioia e il tuo entusiasmo sono sempre stati contagiosi e mi hanno insegnato a guardare la vita con uno sguardo più positivo. Mi considero molto fortunato ad averti conosciuta e ad averti accanto.

Lascio in fondo a questa lista la persona che oggi ha tagliato assieme a me questo traguardo e con la quale ho condiviso la maggior parte dei momenti assieme: Jacopo. Fin dai tempi della scuola ci siamo sempre trovati in sintonia e abbiamo condiviso qualsiasi avventura assieme. Non avrei potuto chiedere un amico più onesto e sincero con cui concludere il mio percorso di studi. Con te ho condiviso davvero tutto: dallo studio ai momenti di svago, dalle difficoltà affrontate assieme alle soddisfazioni raggiunte. La tua presenza è stata per me una costante in questi anni, un punto fermo su cui ho sempre potuto contare e fare riferimento. Sapere di aver percorso anche questo tratto di strada al tuo fianco rende questo traguardo ancora più significativo.

Dedico, infine, questa tesi ai miei genitori, senza i quali non avrei raggiunto questo obiettivo. Mi avete sempre supportato in qualsiasi decisione scelta senza mai pretendere nulla e comprendendomi in tutti i momenti. Sento il vostro affetto e fiducia in ogni nuovo traguardo raggiunto.