Antes de aplicar cualquier algoritmo de genotipado KIR, realizamos un control de calidad exhaustivo de las secuencias NGS. Este paso es crítico, porque la región KIR es altamente repetitiva y sensible a errores, y cualquier problema técnico puede traducirse en genotipos falsos.

Preparación del entorno

Creamos un entorno bioinformático aislado para garantizar reproducibilidad y evitar conflictos de software. Esto asegura que todos los análisis se ejecutan bajo las mismas condiciones, un requisito clave en entornos clínicos.

Bash

```
None
# Crear el entorno
conda create -n bioinfo_reto -c bioconda -c conda-forge fastqc
multiqc -y

# Activar el entorno
conda activate bioinfo_reto

conda install -c bioconda -c conda-forge multiqc -y
```

Bash

```
None
mkdir -p data/fastq    # Aquí pon tus archivos .fastq.gz
mkdir -p results/qc    # Aquí irán los informes de calidad
```

Análisis de calidad individual

Evaluamos cada archivo FASTQ con FastQC para analizar la calidad de las bases, la presencia de adaptadores y posibles sesgos de secuenciación. Este análisis nos permite detectar problemas técnicos antes de cualquier inferencia biológica.

Bash

```
None
# Cambia 'data/fastq/*.fastq.gz' por la ruta donde estén tus
archivos
fastqc -t 8 data/control/*.fastq.gz -o results/qc/
```

Visión global de todas las muestras

Integramos los resultados individuales en un único informe global con MultiQC. Esto nos permite identificar patrones comunes, outliers y problemas sistemáticos en el conjunto de datos, algo esencial cuando trabajamos con cientos de muestras.

Bash

```
None
multiqc results/qc/ -o results/qc/multiqc_report
```

Criterios de decisión

A partir del informe global, definimos criterios objetivos para:

- aceptar muestras de alta calidad,
- aplicar recorte de lecturas si es necesario,
- o descartar datos que comprometerían la fiabilidad del genotipado KIR.

Mensaje clave para el jurado

Antes de predecir genes KIR o respuesta clínica, nos aseguramos de que los datos son técnicamente sólidos. Sin este control, cualquier algoritmo —por sofisticado que sea— produciría resultados poco fiables.”

Nos situamos en la carpeta raíz del proyecto y creamos una estructura dedicada a las referencias biológicas, separando explícitamente la base de datos KIR del resto del pipeline. A continuación, descargamos directamente desde el EBI el archivo FASTA oficial de IPD-KIR, que contiene todas las secuencias genómicas conocidas de los alelos KIR. Trabajar con esta referencia estandarizada garantiza que el índice utilizado para la cuantificación se basa en el catálogo internacionalmente aceptado, lo que es esencial para asegurar la reproducibilidad del análisis y la validez biológica de los resultados obtenidos.

Líneas de código

```shell
cd ~/Desktop/hackaton_lamarato2025
mkdir -p db/kir
cd db/kir
wget https://ftp.ebi.ac.uk/pub/databases/ipd/kir/kir_gen.fasta
-O KIR_alleles.fa
```

Este comando sirve para **extraer y preparar la información de los alelos KIR** a partir del FASTA, creando un fichero de descripción que luego se usa para interpretar los resultados de kallisto. Te lo explico de forma clara y en un solo bloque narrativo, y después te descompongo el comando por partes.

Explicación (en un párrafo, estilo speech)

A partir del archivo FASTA de IPD-KIR, este comando extrae únicamente las cabeceras de las secuencias, que contienen los identificadores de cada alelo. Eliminamos el carácter `>` propio del formato FASTA y guardamos esa información en un fichero de descripciones. Este fichero actúa como un diccionario que permite traducir los identificadores internos utilizados por kallisto a nombres de alelos KIR biológicamente interpretables, haciendo posible convertir la cuantificación en un tipaje comprensible.

Qué hace cada parte del comando

```shell
grep ">" KIR_alleles.fasta
```

- Selecciona solo las líneas que empiezan por `>`.
- En un FASTA, estas líneas son las **cabeceras** de cada secuencia.
- Contienen el identificador del alelo y metadatos.

```shell
sed 's/>//g'
```

- Elimina el carácter `>` de las cabeceras.
- Deja el identificador “limpio”.

```shell
> KIR_descriptions.txt
```

- Redirige la salida a un fichero.
- El resultado es un archivo de texto con una línea por alelo.

Qué contiene `KIR_descriptions.txt` y por qué es clave

Cada línea corresponde a un alelo KIR, por ejemplo:

```
KIR2DL1*0010101

KIR3DL1*0150201

KIR2DS4*00101
```

Este archivo se usa después para:

- cruzar `target_id` de kallisto con alelos reales,
- agrupar alelos por gen,
- construir genotipos interpretables,
- y generar los CSV finales y la *confussion matrix*.

La alternativa: Kallisto (pseudoalineamiento)

En lugar de intentar alinear cada lectura a una posición exacta del genoma, como hacen alineadores clásicos tipo BWA que generan archivos BAM grandes y costosos, Kallisto utiliza un enfoque de **pseudoalineamiento**. En este enfoque, el algoritmo no se pregunta “¿en qué posición exacta cae esta lectura?”, sino “¿con qué alelos es compatible esta lectura?”. A partir de esa compatibilidad, Kallisto estima directamente cuántas lecturas apoyan a cada alelo y produce tablas de cuantificación, sin generar alineamientos explícitos. Este enfoque es especialmente adecuado para regiones como KIR, altamente homólogas, donde muchos reads no se pueden asignar de forma unívoca.

Las ventajas para este reto son claras: en términos de velocidad, Kallisto procesa una muestra en menos de dos minutos frente a los más de diez minutos necesarios con BWA; en consumo de memoria, utiliza muy poca RAM, evitando errores frecuentes como procesos “Killed”; en almacenamiento, genera únicamente archivos de texto de pocos kilobytes en lugar de BAMs de varios gigabytes; y además simplifica enormemente las dependencias, ya que no requiere herramientas como samtools ni librerías problemáticas como libncurses.

Paso 1: Instalación

Instalamos Kallisto directamente desde Bioconda dentro del entorno de trabajo, garantizando una instalación rápida y reproducible:

```shell
conda install -c bioconda kallisto -y
```

Paso 2: Creación del índice (solo una vez)

Antes de analizar muestras, Kallisto necesita construir su propio índice a partir de las secuencias de referencia. En este caso, utilizamos el FASTA oficial de IPD-KIR descargado previamente. Este índice actúa como el “mapa” de todos los alelos KIR contra el que se evaluarán las lecturas:

```shell
cd ~/Desktop/hackaton_lamarato2025

kallisto index -i db/kir/kir_index.idx db/kir/KIR_alleles.fa
```

Este paso se ejecuta una sola vez y puede reutilizarse para todas las muestras del proyecto.

Paso 3: Análisis de una muestra

Una vez creado el índice, el análisis de cada muestra es muy sencillo. Definimos las rutas a los archivos FASTQ paired-end y ejecutamos Kallisto indicando el índice y la carpeta de salida:

```shell
R1="/home/davidlazaro/Desktop/hackaton_lamarato2025/data_controls/data_controls_fq/AMAI-KIR_S4_L001_R1_001.fastq.gz"

R2="/home/davidlazaro/Desktop/hackaton_lamarato2025/data_controls/data_controls_fq/AMAI-KIR_S4_L001_R2_001.fastq.gz"



mkdir -p results/kallisto/AMAI-KIR_S4
```

A partir de aquí, Kallisto cuantifica directamente los alelos KIR presentes en la muestra, produciendo una salida ligera y fácilmente interpretable que se integra con el resto del pipeline para inferir presencia, ausencia y genotipos.

Perfecto. Te lo dejo **redactado como bloque final del discurso**, claro y defendible, integrando **cómo ver los resultados**, **qué significa cada columna** y **por qué este enfoque rediseña la herramienta**. Está listo para decirlo tal cual o ponerlo en una diapositiva.

Paso 4: ¿Cómo ver y entender los resultados?

Una vez finaliza el análisis, Kallisto genera en la carpeta de salida un archivo clave llamado `abundance.tsv`. Este fichero contiene la cuantificación de cada alelo KIR detectado en la muestra. Para inspeccionarlo directamente desde la terminal de forma legible, utilizamos el siguiente comando, que alinea las columnas y muestra solo las primeras filas:

```shell
column -t -s $'\t' results/kallisto/AMAI-KIR_S4/abundance.tsv | head -n 20
```

El archivo contiene varias columnas, pero las más relevantes para nuestro análisis son las siguientes: `target_id`, que identifica el alelo KIR; `length` y `eff_length`, que representan la longitud real y efectiva de la secuencia; `tpm`, que indica la abundancia relativa normalizada; y, especialmente, `est_counts`, que es el número estimado de lecturas que Kallisto asigna a ese alelo. Este valor es la base cuantitativa que utilizamos posteriormente para decidir presencia, ausencia y, combinando ratios entre alelos, inferir genotipos homo u heterocigotos.

¿Por qué este enfoque “rediseña” la herramienta?

Aunque Kallisto se diseñó originalmente para cuantificar expresión génica a partir de RNA-seq, aquí lo estamos utilizando para genotipado de ADN en una región altamente homóloga como KIR. Esta adaptación se basa en un enfoque avanzado conocido como *k-mer based genotyping*, en el que no se fuerza una alineación lineal de cada lectura, sino que se evalúa su compatibilidad con conjuntos de alelos. Al hacerlo, conseguimos evitar sesgos clásicos de alineamiento en regiones repetitivas, multiplicar por diez la velocidad de procesamiento frente a alineadores tradicionales y reducir drásticamente el coste computacional. Esto hace posible analizar cientos de controles sanos en una fracción del tiempo y los recursos que requeriría un pipeline basado en BAMs, sin perder capacidad de discriminación biológica.

Lo que estás viendo en la salida de `python3 script.py` es, básicamente, un "top" de alelos (targets) con mayor soporte según Kallisto; y el `run_kallisto_batch.sh` que has puesto automatiza ese mismo análisis para todas las muestras. Te explico ambos bloques para que puedas justificarlo y, además, te propongo dos mejoras concretas (muy recomendables) para que el batch sea más robusto y más útil para vuestra *confusion matrix*.

---

1) Cómo interpretar la tabla que imprime `script.py`

Has impreso 20 filas con estas columnas:

- **`target_id`**: identificador del *target* en el índice de Kallisto. En tu caso son IDs tipo `KIR:KIR00648`.
  Importante: este ID **no es todavía el nombre del alelo** (p. ej. `KIR2DL1*...`) a menos que tu índice ya lleve esos headers. Normalmente se traduce mediante vuestro `KIR_descriptions.txt` (diccionario `target_id → allele_name`).
- **`length`** y **`eff_length`**: longitud real y longitud efectiva (Kallisto ajusta por sesgos de fragmento / mapeabilidad). En genotipado esto no es el foco principal, pero es parte del modelo de abundancia.
- **`est_counts`**: número "estimado" de lecturas/fracciones asignadas a ese target tras el modelo EM.
  Este es el dato que usáis para decidir "qué alelos dominan" y luego inferir presencia/heterocigosis con ratios.
- **`tpm`**: abundancia normalizada (transcripts per million). Aunque nació para RNA-seq, en vuestro pipeline es útil porque **normaliza por tamaño y profundidad**; para comparar entre muestras suele ser más estable que `est_counts`.

Qué significa la foto que muestras:
hay varios targets con miles de counts y TPM muy altos. Eso es normal cuando el índice contiene muchos alelos similares: Kallisto reparte y concentra señal en un subconjunto que explica mejor el conjunto de k-mers observados.

---

2) Qué hace tu script `run_kallisto_batch.sh` (paso a paso)

```shell
INDEX="db/kir/kir_index.idx"

FASTQ_DIR="data_controls/data_controls_fq"

OUT_BASE="results/kallisto"
```

- Defines el índice (ya construido con `kallisto index`)
- Defines la carpeta donde están los FASTQ
- Defines dónde guardar resultados

```shell
for r1 in $FASTQ_DIR/*_R1_001.fastq.gz; do
```

- Itera por todos los ficheros R1 del directorio
- Asume el patrón de naming `*_R1_001.fastq.gz`

```shell
sample=$(basename $r1 _L001_R1_001.fastq.gz)
```

- Extrae el nombre de muestra eliminando el sufijo `_L001_R1_001.fastq.gz`
- Ejemplo: `AMAI-KIR_S4_L001_R1_001.fastq.gz` → `AMAI-KIR_S4`

```shell
r2="${r1/_R1_/_R2_}"
```

- Deriva la ruta de R2 sustituyendo `_R1_` por `_R2_`
- Esto solo funciona si el naming es consistente y existe R2

```shell
echo "Procesando $sample..."

mkdir -p "$OUT_BASE/$sample"

kallisto quant -i $INDEX -o "$OUT_BASE/$sample" $r1 $r2

done
```

- Crea la carpeta de salida por muestra
- Ejecuta kallisto con paired-end

Resultado: para cada muestra te crea `results/kallisto/<muestra>/abundance.tsv` y otros ficheros.

4) Por qué esto está alineado con vuestra validación

El batch script nos permite procesar **todas las muestras con exactamente el mismo índice y parámetros**, de forma reproducible. Eso es esencial para que la *confussion matrix* sea interpretable: si cambias parámetros entre muestras, introduces sesgo de proceso.

Perfecto. Te explico **cada plot uno por uno**, con **qué mide**, **cómo se interpreta** y **qué mensaje clave puedes transmitir** en un speech. Lo hago en el mismo orden en que suelen presentarse en una validación metodológica.

---

1) Accuracy por muestra (Presencia/Ausencia)

Qué muestra
Cada barra corresponde a una **muestra individual**. El valor es la proporción de genes KIR en los que tu método (RESULTADO) coincide con el método de referencia (TIPATGES) en términos de **presencia o ausencia**, sin entrar en alelos concretos.

Cómo se calcula
[
\text{accuracy} = \frac{\text{genes correctamente clasificados}}{\text{total de genes evaluados}}
]

Cómo interpretarlo

- Muestras con accuracy ≈ **1.0**: concordancia casi perfecta.
- Muestras alrededor de **0.8–0.9**: buena concordancia, con algunos errores puntuales.
- Muestras claramente más bajas (por ejemplo KAS011_KIR): posibles problemas de:
    - baja cobertura,
    - calidad del FASTQ,
    - genotipo complejo,
    - o limitaciones del índice.

Mensaje clave

> "El rendimiento es consistente entre la mayoría de muestras, con algunos outliers que identificamos claramente para análisis posterior."

---

2) Métricas por gen (Presencia/Ausencia)

(Panel con *accuracy*, *PPV*, *sensitivity* y *specificity*)

Este conjunto de gráficos responde a la pregunta:
¿En qué genes funciona mejor o peor el método?

2.1 Accuracy por gen

Qué mide
Proporción total de aciertos (presentes + ausentes) para cada gen.

Interpretación

- Genes con accuracy alta (>0.9): muy fiables.
- Genes con accuracy baja (ej. 2DS1, 2DL5A/B): problemáticos.

Por qué pasa
Normalmente:

- mayor homología,
- CNV,
- pseudogenes,
- menor representación en el índice.

2.2 PPV (Positive Predictive Value)

Qué mide
Cuando el método dice "presente", **qué proporción es correcta**.

Interpretación

- PPV alto → pocas falsas presencias.
- PPV bajo → tendencia a **sobrellamar** el gen.

Mensaje

> “Algunos genes tienden a falsos positivos, lo que nos indica dónde ajustar umbrales.”

2.3 Sensitivity (Recall)

Qué mide
Capacidad de detectar un gen cuando realmente está presente.

Interpretación

- Sensitivity alta → pocos falsos negativos.
- Sensitivity baja → el método se “pierde” genes reales.

2.4 Specificity

Qué mide
Capacidad de detectar correctamente la ausencia de un gen.

Interpretación

- Specificity alta → no inventa genes.
- Specificity baja → confunde ausencia con presencia.

Mensaje global del panel

> “El rendimiento depende del gen, lo cual es esperable en la región KIR, pero nos permite identificar de forma precisa qué genes requieren ajustes específicos.”

---

3) Heatmap de discrepancias (1 = distinto)

Qué muestra

- Filas: muestras
- Columnas: genes

- Azul (0): acuerdo TIPATGES–RESULTADO
- Rojo (1): discrepancia

Por qué es potente
Permite detectar **patrones**, no solo errores individuales:

- columnas muy rojas → genes sistemáticamente problemáticos
- filas muy rojas → muestras problemáticas
- clusters → perfiles similares de error

Mensaje clave

> "Los errores no son aleatorios: se concentran en genes y muestras concretas, lo que indica causas técnicas identificables y corregibles."

---

4) Exact match por gen (excluyendo '+')

Qué evalúa
Aquí ya no es presencia/ausencia, sino **coincidencia exacta de alelos**
(se excluyen casos ambiguos tipo '+').

Interpretación

- Valores bajos son esperables:
  el alelo exacto es mucho más exigente que presencia/ausencia.
- Genes con algo de exact match (ej. 2DL5B, 2DS4):
  → potencial real de tipado alélico.
- Genes con 0:
  → homología extrema o cobertura insuficiente.

Mensaje clave

> "Aunque el objetivo principal es presencia/ausencia, algunos genes muestran capacidad real de tipado alélico exacto."

---

5) Confusion matrix global (Presencia/Ausencia)

Qué resume
Es el **resumen final global** de todo el análisis.

Ejes

- Y: verdad (TIPATGES)
- X: predicción (RESULTADO)

Celdas

- **True Positive (21)**: presente y bien detectado
- **False Negative (307)**: presente pero no detectado
- **False Positive (43)**: detectado pero no estaba
- **True Negative (109)**: ausencia bien detectada

Cómo leerla

- El método es **conservador**:
    - tiende más a falsos negativos que a falsos positivos
- Esto es preferible en un contexto clínico/exploratorio:
  mejor no inventar genes que están ausentes.

Mensaje clave

> "Nuestro método prioriza especificidad sobre sensibilidad, lo que reduce falsos positivos y es adecuado como primer cribado."

🧠 Mensaje global para cerrar el bloque

> "La validación muestra un rendimiento sólido en presencia/ausencia, con variabilidad dependiente del gen y de la muestra. Los errores no son aleatorios, lo que abre la puerta a optimizaciones dirigidas del índice y de los umbrales."