Supplementary Table 1: GA4GH recommendations for best practices for germline variant call benchmarking

Benchmark sets	Use benchmark sets with both high-confidence variant calls as well as high-confidence regions (e.g., from GIAB or Platinum Genomes). Record versions of the benchmark sets.
Stringency of variant comparison	Determine whether it is important that the genotypes match exactly, only the alleles match, or the call just needs to be near the true variant. For example, if you confirm and/or manually curate all variants to ensure you have the correct allele and genotype, then local matching may be sufficient. While the default TP, FP and FN require genotype and allele matching, the additional metrics FP.GT and FP.AL output by the GA4GH tools enable users to calculate performance at different stringencies.
Variant comparison tools	Use sophisticated variant comparison tools such as vcfeval,[10] xcmp,[20] or varmatch[11] that are able to determine if different representations of the same variant are consistent with the benchmark call (examples in Fig. 1; comparison tools in Supplementary Note 2). Subsetting by high-confidence regions and, if desired, targeted regions, should only be done after comparison to avoid problems comparing variants with different representations.
Manual curation	Manually curate alignments, ideally from multiple data types, around at least a subset of putative false positive and false negative calls in order to ensure they are truly errors in the user's callset and to understand the cause(s) of errors. Report back to benchmark set developers any potential errors found in the benchmark set (e.g., using https://goo.gl/forms/ECbjHY7nhz0hrCR52 for GIAB or https://github.com/Illumina/PlatinumGenomes/issues/new for PG).
Interpret-ation of metrics	All performance metrics should only be interpreted with respect to the limitations of the variants and regions in the benchmark set. Variant types not present in the benchmark set, and variants detected outside of high-confident regions, such as those in repetitive or difficult-to-map regions, will remain unassessed. Hence, when comparing methods, note that method A may perform better in the high-confidence regions, but method B may perform better for more difficult variants outside of these regions. To aid in the accurate interpretation of results, report the type and number of variants covered by the benchmarking set, the size of the confident regions, and the fraction of "not assessed" query calls that fall outside of these regions. Performance metrics are likely to be lower for more difficult variant types and regions that are not fully represented in the benchmark set, such as those in repetitive or difficult-to-map regions. We recommend limiting the reportable range to variants and regions that can be confidently assessed with a given truth set, because accuracy will often be lower for the more challenging variants not included in the high-confidence variants and regions.
Stratifi-cation	Performance results should be stratified by variant type. Stratification by genomic region should also be considered to gain additional insights into strengths and limitations of the sequencing pipeline, as it can highlight regions that are not sufficiently represented. Stratification should only be done after comparison to avoid problems comparing variants with different representations.
Confidence Intervals	Confidence (or credible) intervals for performance metrics such as precision and recall should be calculated (e.g., using https://github.com/Illumina/happyCompare or R binconf). This is particularly critical for the smaller numbers of variants found when benchmarking targeted assays and/or less common stratified variant types and regions.
Additional bench-marking approaches	We recommend using other benchmarking approaches in addition to those discussed in this paper to understand performance of a pipeline, including: ● Confirming results found in samples over time ● Synthetic DNA spike-ins with challenging and common clinically relevant variants ● Engineering variants into cell lines ● Finding existing samples with challenging and common clinically relevant variants ● Simulation methods, such as read simulators, adding variants into real reads, and modifying the reference ● Run-specific metrics such as base quality score distributions, coverage distributions, etc. can also be useful to identify outlier runs

Supplementary Table 2: Definitions and formulas for performance metrics output by GA4GH tools. Bold metrics are "Tier 1" metrics displayed by default, and others are "Tier 2" metrics output but optionally displayed.

Metric	Common Name(s)	Definition	Formula
TRUTH.TP	**True positives (Truth)**	**Number of truth calls for which there is a query call that is consistent with the truth call and its genotype**	
QUERY.TP	**True positives (Query)**	**Number of query calls for which there is a truth call that is consistent with the query call and its genotype. This can differ from TRUTH.TP if complex changes are represented as a single change in TRUTH.TP and as multiple primitive SNVs and indels in QUERY.TP, or vice versa.**	
TRUTH.FN	**False negatives**	**Number of truth calls for which there is no query call that is consistent with the truth call and its genotype**	
QUERY.FP	**False positives**	**Number of query calls for which there is no truth call that is consistent with the query call and its genotype.**	
QUERY. UNK	Number of unknown variant calls	The number of query variant calls not inside confident regions of the truth dataset	
QUERY. TOTAL		**Total number of query calls**	**QUERY.TP + QUERY.FP + QUERY.UNK**
TRUTH. TOTAL		Total number of truth calls	TRUTH.TP + TRUTH.FN
METRIC. Recall	**Recall, Sensitivity**	**Fraction of truth calls that are consistent with a query allele and genotype call within the confident regions**	**TRUTH.TP / (TRUTH.TP + TRUTH.FN)**
METRIC. Precision	**Precision, Positive predictive value**	**Fraction of query calls that are consistent with a truth allele and genotype call within the confident regions**	**QUERY.TP / (QUERY.TP + QUERY.FP)**
METRIC. Frac_NA	**Fraction not assessed**	**Fraction of query calls that are outside the confident regions of the truthset (and which could not be assessed in this benchmarking run)**	**QUERY.UNK / QUERY.TOTAL**
F-Score	**F1 Score**	**The harmonic mean between recall and precision**	**2 * METRIC.Recall* METRIC.Precision / (METRIC.Recall + METRIC.Precision)**
FP.GT	Genotype errors	This is the number of query variants with an incorrect genotype, but the correct allele (e.g., when the query GT is 1/1 and truth GT is 0/1)	
FP.AL	Allele errors	The number of query variant calls which could not be matched by genotype or by alleles, but which have a truth variant call within a specified distance	

Supplementary Note 1: Variant Representation Challenges

Variant representation differences broadly fall into the following categories.

1. *Reference-trimming of alleles*: Variant alleles may include reference bases at the beginning or at the end. This is required to represent insertions in a VCF file, which always include a single padding base (the reference base just before the insertion), and is also used by most variant calling methods to represent deletions. Some methods add reference padding to explicitly assert surrounding bases have been observed to be homozygous reference.
2. *Left-shifting and right-shifting of alleles*: variant calls in repetitive sequence may have different possible starting positions after trimming. The simplest example of this scenario is change in length in a homopolymer: this can be represented as an insertion/deletion which may be anchored at any position within the homopolymer (Fig 2a). Interestingly, the unstated convention for VCF is that variants are left-aligned, whereas HGVS guidelines state that variants should be right-aligned in the context of the transcript (i.e., represent the 3' most possibility), which may be either left or right aligned in the genomic context depending on which strand the gene is encoded on. More complex repeats lead to additional complexity.[9–11]
3. *Allele decomposition and phasing*: Complex alleles may be represented as a single VCF record, or split into multiple records with different starting positions. An insertion can often be represented as one large insertion or multiple smaller insertions (Fig 2b). Similarly, multi-nucleotide polymorphisms (MNPs) can be represented either as a single MNP record, or alternatively as separate SNVs (Fig 2c). The MNP representation may be chosen since it explicitly encodes variant phasing and makes it clear that two SNVs have occurred on the same haplotype, which can have a significant impact on clinical interpretation, as illustrated in Supplementary Figure 1. In the SNV representation the same information may be encoded using the PS format field in the VCF file, but the practical implementation varies between different variant calling methods. Also, although including local phasing is strongly encouraged, phasing information may not be present in the VCF file, in which case we can match the MNP to the SNVs by decomposing it, but we cannot recompose the SNVs into the MNP if they are unphased and heterozygous.
4. *Generic ambiguous alignment*: Cases 1 and 2 are specific examples for ambiguous alignments between reference sequence and the observed haplotype in a sample. However, these are not the only case where different sequence alignments with the same maximum score may exist. In low-complexity sequence, more difficult scenarios may arise. One example is the case where we have a SNV adjacent to a deletion. The SNV may be called either at the start or at the end of the deletion, and the choice is up to the variant caller or the aligner. An example is shown in Fig. 2d. Complex variants in repetitive regions often cannot be "normalized" (i.e., converted into a standard representation) by any existing tools, so that sophisticated variant comparison tools like those developed in this work are necessary.

Supplementary Note 2: Comparison tools

The matching methods have been implemented in different variant comparison engines which can be used as part of the GA4GH benchmarking workflow. These methods are:

- *scmp-distancebased*: this will match variants by location only. Any variant in the query that has a truth variant nearby within a specified distance.
- *scmp-somatic*: this mode is intended to be used with Tumor/Normal VCF files. Variants in the query will be matched to truth variants if the same normalized alleles are found nearby. Genotypes are ignored, and multi-sample VCF files will be collapsed into a single column with a pre-specified genotype. Alleles are split out into one allele per VCF line.
- *RTG Tools vcfeval* (see below for more detailed instructions): this is a comparison that is similar to xcmp, but which uses an optimization method to determine TP/FP status on a per-variant level (rather than xcmp's per-superlocus level). This method does not use phasing information when it is present in the input VCF file and produces genotype matches. Vcfeval can also perform allele matching when using the --squash-ploidy option, dealing appropriately with variant representation issues, which is useful for somatic callset benchmarking.
- *xcmp* (hap.py's default comparison engine): this method will assume that both input samples are diploid / human samples. Matching is performed on a haplotype level: xcmp enumerates all possible haplotypes that may be described by truth and query within a small superlocus. If matching pairs of haplotype sequences are found, xcmp will convert all variants within the surrounding superlocus into TPs. Xcmp also recognizes direct genotype or allele matches / mismatches. Global phasing information is used to restrict haplotype enumeration for phased genotype matching, but PS phasing is not supported.

When benchmarking, these variant representation differences can also give rise to different notions of giving partial credit for variant calls, such as when a method calls only one SNV in a multi-nucleotide variant (MNV, defined as multiple, adjacent SNVs). When assigning TP/FP/FN status on a per-VCF-record basis, a variant caller that chooses to represent calls using single MNP records would not get credit for calling this SNV correctly since the overall MNP record does not reproduce the correct haplotype. Another example would be phasing switch-errors: a choice needs to be made whether to use phasing-aware benchmarking for a particular evaluation. Handling these cases is important since adding phasing information provides additional information to the users of a variant caller, but may lead to FPs / FNs when running a benchmarking comparison when comparing to a method which does not provide phasing information and outputs all alleles in decomposed form for maximum credit in the benchmarking comparison. Our tools attempt to give partial credit when possible, and we generally recommend using vcfeval as the comparator to provide the most partial matches.

Supplementary Note 3: Benchmarking VCF file formats

Intermediate VCF Files

The intermediate VCF file is produced by a comparison engine (like xcmp / vcfeval / xcmp in hap.py).

A comparison engine should

- assess for each call in the truth and in the query whether this call is considered a match or mismatch, and ideally output if the allele matches but the genotype does not (FP.GT) or if the wrong allele is called near a true allele (FP.AL)
- output corresponding labels:
    - True Positive / `TP`: present in both truth and query
    - False Positive / `FP`: present only in the query
    - False Negative / `FN`: present only in the truth
    - Not-assessed / `N`: call was not assigned a match status
- (optionally) output additional information about each decision

Note that the comparison engine should output the FP/TP/FN/N labels separately for truth and for query variants. For simple comparison types which do not attempt to reconcile different variant representations, the assigned type for truth and query might be the same. However, more sophisticated methods (e.g. vcfeval) will be able to type truth and query variants separately.

The N label may be applied for a variety of reasons, which may be specific to the comparison engine. For example, a comparison engine might not assess input calls which had non-PASS FILTER fields, or may choose to ignore half-calls. Alternatively an engine may find that some call regions are too complex to confidently assess. Additional information may be included in the BI annotation.

A comparison engine should also preserve input INFO / FORMAT annotations to the largest degree possible (depending on the variant processing it does).

Required VCF Fields

The intermediate VCF must have two columns named TRUTH and QUERY, with these FORMAT annotations:

```
##FORMAT=<ID=BK,Number=1,Type=String,Description="Sub-type for decision
(match/mismatch type)">
```

The value of BK specifies the class of match for each variant record:

1. `.`: *missing* = no match at any level tested by the comparison tool
2. `lm`: *local match* = the truth/query variant is nearby a variant in the query/truth -- if the tool outputs such match types, it should annotate the VCF header with the definition of local matches (e.g. match within a fixed window, or within the same superlocus).

3. `am`: *almatch* = the variant forms (part of) an allele match (independent of representation, i.e. one-sided haplotype match)
4. `gm`: *gtmatch* = diploid haplotypes (and genotypes) were resolved to be the same (independent of representation)

Based on the values in BK, comparison tools must assign a decision for each variant call that assigns true/false positive/negative status. This status is output in the BD format field:

```
##FORMAT=<ID=BD,Number=1,Type=String,Description="Decision for call (TP/FP/FN/N)">
```

The mapping of combinations of BK values to BD could vary for different comparison stringencies, as discussed in the Variant Counting section. Our current tools require genotypes to match for TP to be in the BD field.

Local Matching

Find all variants in the query where another variant was seen nearby in the truth.

BK	BD (Truth)	BD (Query)
.	FN	FP
lm/am/gm	TP	TP

Allele Matching

Test allele-level concordance between truth and query.

BK	BD (Truth)	BD (Query)
./lm	FN	FP
am/gm	TP	TP

Genotype Matching

Test genotype concordance between truth and query.

BK	BD (Truth)	BD (Query)
./lm	FN	FP
am	FN	FP (and FP.GT)
gm	TP	TP

Additional VCF Fields

Comparison engines may have engine-specific status information that is useful to present in the output (for example, error status, specific match sub-algorithm used, etc). This can be recorded in the optional BI annotation:

```
##FORMAT=<ID=BI,Number=1,Type=String,Description="Additional match status information">
```

We allow for comparison engines to transform the input variants for more granular accounting / comparison. To facilitate ROC creation based on such a processed variant file, we define the following FORMAT annotation to pass on a variant quality score obtained from the input variants:

```
##FORMAT=<ID=QQ,Number=1,Type=Float,Description="Variant quality for ROC creation.">
```

Since we aim to benchmark variant calls independently of their representation, we also define *superloci*. A *superlocus* is a set of variant calls that intends to fully describe the variation within a contiguous reference region that may contain complex variation with different representations. In order to group variants into blocks by superlocus, we introduce the following INFO annotation that allows us to assign a benchmarking superlocus ID to each variant record.

```
##INFO=<ID=BS,Number=1,Type=Integer,Description="Benchmarking superlocus ID for these variants.">
```

In downstream tools, this may e.g. be used to count with superlocus granularity.

Final Output VCF Files

The output VCF file is similar to the intermediate VCF file. We add one additional variant classification: in addition to FP/FN/TP, each variant call can also be assigned the status UNK for unknown / outside the regions which the truthset covers.

A simple definition for UNK variants is as follows: We call any variant unknown if it BD == FP and BK == miss and if the variant is outside the confident call regions of the truth set. If the truthset does not give confident call regions, no UNK variants are output.

We introduce the following additional fields:

```
##INFO=<ID=Regions,Number=.,Type=String,Description="Tags for confident / stratification regions.">
##FORMAT=<ID=BVT,Number=1,Type=String,Description="High-level variant type in truth/query (SNV|INDEL|COMPLEX).">
##FORMAT=<ID=BLT,Number=1,Type=String,Description="High-level location type in truth/query (het|hom|hetalt|halfcall|multiallelic|nocall).">
```

Optional: variant types seen and counted for this record.

```
##INFO=<ID=VTC,Number=.,Type=String,Description="Variant types used for counting.">
```

Supplementary Note 4: Variant types for stratification

Counts and statistics are calculated for the following subsets of variants:

Type	Description
SNV	SNV or MNP variants. We count single nucleotides that have changed
INDEL	Indels and complex variants

Hap.py (and qfy.py, which is the part of the hap.py that counts variants) also computes counts for subtypes and observed genotypes in the above two categories.

Subtype	Description
*	Aggregate numbers for all subtypes
ti or tv	Transitions and transversions for SNVs
I1_5	Insertions of length 1-5
I6_15	Insertions of length 6-15
I16_PLUS	Insertions of length 16 or more
D1_5	Deletions of length 1-5
D6_15	Deletions of length 6-15
D16_PLUS	Deletions of length 16 or more
C1_5	Complex variants of length 1-5
C6_15	Complex variants of length 6-15
C16_PLUS	Complex variants of length 16 or more

Genotype	Description
*	Aggregate numbers for all genotypes
het	only heterozygous variant calls (0/1 or similar genotypes)
homalt	only homozygous alternative variant calls (1/1 or similar genotypes)
hetalt	only heterozygous alternative variant calls (1/2 or similar genotypes)

Note that currently the granularity of counting is on a per-VCF-record level. In complex/high-variability regions, the classifications above might become inaccurate due to nearby variants (e.g. an insertion with a close-by SNV would be more accurately classified as "complex" if the SNV and the insertion occur on the same haplotype).

For each variant stratification subset, the GA4GH benchmarking workflow outputs a set of stratification columns, followed by metrics columns. Stratification columns may contain the placeholder "*" value, which indicates that values in this row are aggregated over all possible values of the column. In case of a subtype, this means that the counts have been summed across all subtypes. In case of QQ, it means that the counts correspond to all variants rather than only ones below a QQ threshold.

Stratification Column	**Description**
Type	Variant type (SNV / INDEL)
Subtype	Variant Subtype (ti/tv/indel length, see above)
Subset	Subset of the genome/stratification region
Filter	Variant filters: PASS, SEL, ALL, or a particular filter from the query VCF
Genotype	Genotype of benchmarked variants (het / homalt / hetalt)
QQ.Field	Which field from the original VCF was used to produce QQ values in truth and query
QQ	QQ threshold for ROC values

Metric Column	Description
METRIC.Recall	Recall for truth variant representation = TRUTH.TP / (TRUTH.TP + TRUTH.FN)
METRIC.Precision	Precision of query variants = QUERY.TP / (QUERY.TP + QUERY.FP)
METRIC.Frac_NA	Fraction of non-assessed query calls = QUERY.UNK / QUERY.TOTAL
METRIC.F1_Score	Harmonic mean of precision and recall = 2*METRIC.Recall*Metric.Precision/(METRIC.Recall + METRIC.Precision)
TRUTH.TOTAL	Total number of truth variants
TRUTH.TP	Number of true-positive calls in truth representation
TRUTH.FN	Number of false-negative calls = calls in truth without matching query call
QUERY.TOTAL	Total number of query calls
QUERY.TP	Number of true-positive calls in query representation
QUERY.FP	Number of false-positive calls in the query file (extra query calls in truth bed file)
QUERY.UNK	Number of query calls outside the confident regions
FP.gt	Number of genotype mismatches (alleles match, but different zygosity)
FP.al	Number of allele mismatches (variants matched locally but not by alleles)
TRUTH.TOTAL.TiTv_ratio	Transition / Transversion ratio for all truth variants
TRUTH.TOTAL.het_hom_ratio	Het/Hom ratio for all truth variants
TRUTH.FN.TiTv_ratio	Transition / Transversion ratio for false-negative variants
TRUTH.FN.het_hom_ratio	Het/Hom ratio for false-negative variants
TRUTH.TP.TiTv_ratio	Transition / Transversion ratio for true positive variants
TRUTH.TP.het_hom_ratio	Het/Hom ratio for true positive variants
QUERY.FP.TiTv_ratio	Transition / Transversion ratio for false positive variants
QUERY.FP.het_hom_ratio	Het/Hom ratio for false-positive variants
QUERY.TOTAL.TiTv_ratio	Transition / Transversion ratio for all query variants
QUERY.TOTAL.het_hom_ratio	Het/Hom ratio for all query variants
QUERY.TP.TiTv_ratio	Transition / Transversion ratio for true positive variants (query representation)
QUERY.TP.het_hom_ratio	Het/Hom ratio for true positive variants (query representation)
QUERY.UNK.TiTv_ratio	Transition / Transversion ratio for unknown variants
QUERY.UNK.het_hom_ratio	Het/Hom ratio for unknown variants
Subset.Size	When using stratification regions, the number of nucleotides in the region
Subset.IS_CONF.Size	This gives the number of confident bases (-f regions) in the region

Supplementary Note 5: Merging GIAB and PG Truth Calls for NA12878

Platinum Genomes v2016.1 (PG) and NIST Genome in a Bottle v3.3.2 (GIAB) calls were combined through a k-mer validation process adapted from that used in [PG 2017]. The steps in this procedure are as follows:

1. Use hap.py to compare both NA12878 truth set VCFs and identify those records exclusive to GIAB (i.e. 'false positives' in the GIAB query)
2. Merge these GIAB-exclusive calls with the PG set to produce a combined VCF of candidate records for validation
3. For fully-phased loci:
    a. Induce k-mers containing local haplotype sequence of 51 bp centered on each record of the merged VCF
    b. Count exact k-mer matches in aligned reads in the region (+/- 400 bp) of the VCF record for each haplotype sequence
    c. For each haplotype, sum the counts over expected inherited haplotypes in the lower pedigree and divide by the number of expected inheritances to get a normalised k-mer score (KM)
    d. Take the minimum KM of both NA12878 haplotypes in the phased record and filter those where the minimum KM is less than 1
4. For loci spanning one or more unphased records:
    a. Generate all possible k-mer sequences considering any phased or unphased records in the merged VCF within the 51 bp window
    b. Count exact matches in aligned sequence reads for all k-mers, again in the aligned reads surrounding the site (+/- 400 bp). Those k-mers with >=2 counts in NA12878 are taken forward as candidate haplotypes
    c. Assign a k-mer to a PG haplotype label (relative to NA12878) based on which assignment maximises the normalised k-mer score (KM, see above) and subsequently filter any haplotype with KM < 1
    d. If a single pair of haplotypes are able to form a complementary diplotype, this is a validated and phased record. If there are more than one valid haplotype pairs (as may happen at short tandem repeats) the record cannot be validated and is discarded
5. Newly-validated calls are added to the PG confidence tracks

Steps 1 and 2 in the above procedure sidestep the problem of conflicting sites between the two call sets. An improved approach would be to consolidate and arbitrate conflicts, else filtering ambiguous sites where the true state cannot be determined.

The k-mer validation method could also be applied so as to graft PG-exclusive calls onto the GIAB truth set, however many of these PG exclusive calls will fall outside GIAB confident regions and as such would benefit from a more sophisticated method capable of integrating confident regions.