Supplementary Note 1| Analysis of unintended editing types.

Analysis of our screening data (library 1) with CRISPResso2 analysis revealed that most unintended mutations occurred at the positions after the RTT end in the form of scaffold insertions, similar to what was described in Anzalone et al. 2019[1] (**Supplementary Fig. 2a,b**). As expected for scaffold integrations, most of these mutations occurred together with the intended edit (leading to "imperfect editing"; **Supplementary Fig. 2c,d**). For every library/experiment in our study, editing rates (intended and unintended) were corrected by the corresponding value in the untreated control condition, as described in the Methods section. Detailed visualizations of CRISPResso2 read distribution for 4 pegRNA examples (2 replacement edits and 1 insertion and 1 deletion) are visualized in **Supplementary Fig. 2e-h**.

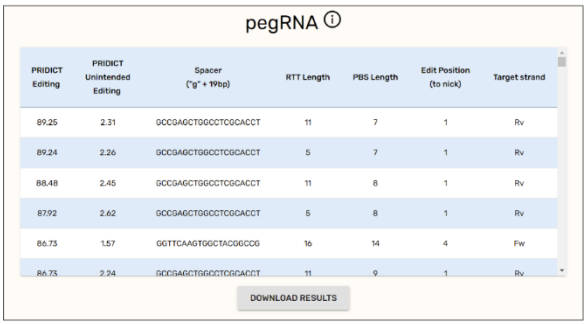
Supplementary Note 2 | Analysis of chromatin effect on intended editing rates.

To analyze the influence of chromatin characteristics on editing efficiency, we analyzed editing efficiencies on endogenous sites vs. the same target sequence randomly integrated via lentivirus (analysis of a pool of cells assures that influences on chromatin by integrated sequences are averaged). We found that open chromatin (ATAC-seq, DNAse-seq) as well as H3K4me3, H3K27me3 and H3K36me3 correlated with a lower difference of editing on integrated vs. endogenous sites at our 15 endogenous loci (**Supplementary Fig. 17a-c**). Since the differences in ATAC-seq and DNAse-seq profiles at the 15 loci we targeted were rather small, we further performed prime editing on 10 additional loci (total of 19 pegRNAs with PRIDICT scores > 50), which were previously shown to be located in open or closed chromatin in HEK293T[25] (**Supplementary Fig. 17d**). Also at these loci we observed a positive effect of open chromatin, H3K4me3 and H3K27me3 on editing efficiencies, while H3K9me3 and H3K36me3 differed in their correlation compared to the previous 15 loci (**Supplementary Fig. 17e,f**). Notably, the positive effect of open chromatin is in line with previous studies, which demonstrated that open chromatin has a positive influence on Cas9 binding and cutting[30] (**Supplementary Fig. 17b**). However, none of the analyzed chromatin characteristics were sufficient to explain why certain pegRNAs with high PRIDICT scores only led to low editing at endogenous loci in HEK293T or K562 cells (**Supplementary Fig. 17g-j**). This suggests that a more detailed analysis of the influence of different chromatin marks on prime editing would be interesting.

Supplementary Note 3 | Description of PRIDICT online workflow.

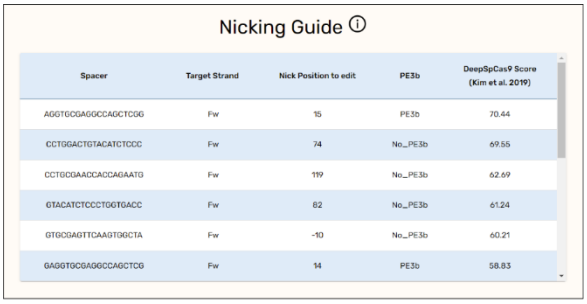
After accessing PRIDICT via a web browser (pridict.it), the user gets presented with the starting page. A description of how to enter the sequence of the desired mutation is described in the grey box. The mutation has to be flanked by 100 bp up- and 100 bp downstream of the mutation for sufficient sequence context. The mutation has to be surrounded by brackets (), and replacement edits can be separated by "/" to distinguish original/edited bases. Deletions should be marked with a "-" in front of the bases to be deleted, while insertions should be marked with a "+". Examples can be inserted by selecting one of the editing types above the grey box.

After entering the sequence and clicking on "PRIDICT! Design pegRNA", the results are displayed after a brief waiting period. The results under the table "pegRNA" are sorted by PRIDICT (intended) editing score, and the unintended editing score is displayed next to it. Further characteristics of the different pegRNAs (Spacer, RTT length, PBS length, Edit position, and Target strand) are listed as well.

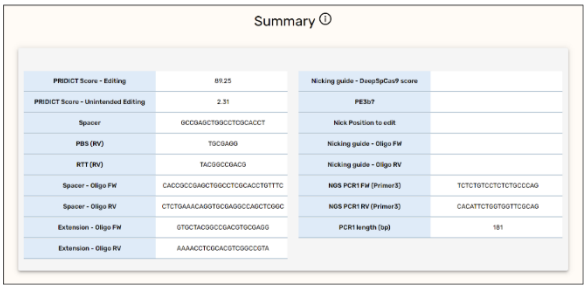


Results can be downloaded via "Download Results", which includes 3 tables listing pegRNA predictions, nicking guides (not part of PRIDICT predictions), and primer designs by Primer3.

When selecting a pegRNA by clicking on it, a new table will be visualized, which lists optional nicking guides for PE3 editing, ranked by DeepSpCas9 score (Kim et al. 2019). Nicking guides are not part of the PRIDICT score or manuscript but are listed for increased user experience.



Finally, a summary table is shown, including the pegRNA (and, if selected, nicking guide) and oligos that can be used for cloning the respective gRNA. Furthermore, a primer pair is listed that can be used for NGS PCR based on the Primer3 design algorithm (if primers with minimal requirements are available).



Supplementary Table 1 | Feature overview. List of features used in training baseline models and PRIDICT deep model.

Feature name	Feature description	Used for baseline models	Used for PRIDICT
baseafter	Base 3' to edit	x	
basebefore	Base 5' to edit	x	
basebeforeafter	Base 5' and 3' to edit	x	
Correction_Length	Length of the correction edit	x	x
Correction_Type	Replacement, Insertion, or Deletion	x	x
deepcas9	DeepSpCas9 value from Kim et al. 2019	x	
edited_base_mt	Melting temperature of the edited sequence stretch (e.g., mt of T if the edit is GtoT)	x	x
edited_base_mt_nan	Is 1 if edited_base length is 0bp (deletions), otherwise 0.	x	x
edited_GC_content	GC content of edited sequence stretch	x	
Editing_Position	Editing position starting from the first base of RT template (RTT)	x	
Extension_GC_content	GC content of extension sequence	x	
extensionmt	Melting temperature of extension sequence	x	x
flapbaseafter	Base on target DNA 3' to edited flap-end	x	
flapbasebefore	Base on target DNA 5' to edited flap-end	x	
MFE_extension	Minimum free energy of RNA (ViennaRNA Package 2) of the extension part	x	x
MFE_extension_scaffold	Minimum free energy of RNA (ViennaRNA Package 2) of the scaffold+extension part	x	x
MFE_pbs	Minimum free energy of RNA (ViennaRNA Package 2) of the PBS part	x	x
MFE_spacer	Minimum free energy of RNA (ViennaRNA Package 2) of the spacer sequence	x	x
MFE_spacer_extension_scaffold	Minimum free energy of RNA (ViennaRNA Package 2) of the full pegRNA (spacer+scaffold+extension)	x	x
MFE_spacer_scaffold	Minimum free energy of RNA (ViennaRNA Package 2) of spacer+scaffold part	x	x
MFE_rt	Minimum free energy of RNA (ViennaRNA Package 2) of RTT part	x	x
nickbaseafter	Base on target DNA 3' to the DNA nick	x	
nickbasebefore	Base on target DNA 5' to the DNA nick	x	
nickbasebeforeafter	Combination of nickbasebefore and nickbaseafter	x	
original_base_mt	Melting temperature of the original sequence stretch (e.g., mt of G if the edit is GtoT)	x	x
original_base_mt_nan	Is 1 if original_base length is 0bp (insertions), otherwise 0.	x	x
original_GC_content	GC content of original sequence stretch	x	
PBS_GC_content	GC content of PBS sequence	x	
PBSlength	Length of PBS sequence (bp)		x
PBSlocation	Location of PBS (relative to position 10bp upstream/5' of protospacer)		x
PBSmt	Melting temperature of PBS sequence	x	x
polyavalues	Max number of consecutive A bases in spacer/extension region	x	

Feature name	Feature description	Used for baseline models	Used for PRIDICT
polycvalues	Max number of consecutive C bases in spacer/extension	x	
polygvalues	Max number of consecutive G bases in spacer/extension	x	
polytvalues	Max number of consecutive T bases in spacer/extension	x	
Proto_GC_content	GC content of spacer sequence	x	
protobase_1	Base at position 1 of protospacer	x	
protobase_10	Base at position 10 of protospacer	x	
protobase_11	Base at position 11 of protospacer	x	
protobase_12	Base at position 12 of protospacer	x	
protobase_13	Base at position 13 of protospacer	x	
protobase_14	Base at position 14 of protospacer	x	
protobase_15	Base at position 15 of protospacer	x	
protobase_16	Base at position 16 of protospacer	x	
protobase_17	Base at position 17 of protospacer	x	
protobase_18	Base at position 18 of protospacer	x	
protobase_19	Base at position 19 of protospacer	x	
protobase_2	Base at position 2 of protospacer	x	
protobase_3	Base at position 3 of protospacer	x	
protobase_4	Base at position 4 of protospacer	x	
protobase_5	Base at position 5 of protospacer	x	
protobase_6	Base at position 6 of protospacer	x	
protobase_7	Base at position 7 of protospacer	x	
protobase_8	Base at position 8 of protospacer	x	
protobase_9	Base at position 9 of protospacer	x	
protospacerlocation_only_initial	Location of protospacer in original sequence (relative to position 10bp upstream/5' of protospacer)		x
protospacermt	Melting temperature of the protospacer sequence	x	x
RT_GC_content	GC content of RTT	x	
RT_initial_location	Location of RTT in original sequence (relative to position 10bp upstream/5' of protospacer)		x
RT_mutated_location	Location of RTT in mutated sequence (relative to position 10bp upstream/5' of protospacer)		x
RTlength	Length of RTT (bp)	x	x
RTmt	Melting temperature of RTT	x	
RToverhanglength	Length of the sequence after the edit on the RTT	x	x
RToverhangmatches	Number of matches of RTT overhang within a window of 15bp downstream of edit	x	x
RToverhangmt	Melting temperature of RTT overhang sequence	x	x
wide_initial_target	Sequence before editing, starting 10bp upstream of protospacer and ending after 99 base pairs.		x
wide_mutated_target	Sequence after editing, starting 10bp upstream of protospacer and ending after 99 base pairs.		x

Supplementary Table 2 | Comparison of study setups. Comparison of spacer type, editing duration, and selection type between this study and external datasets.

Study	Spacer type	Scaffold	Editing time (days)	Selection type
Anzalone et al. 2019	G (matched) + 19 bp spacer (for PE2 edits analyzed in this study (Mathis et al.))	original	3	-
Kim et al. 2021	G/g (matched/mismatched) + 19 bp spacer	original	4.8	Blasticidin
This study (Mathis et al.)	G/g (matched/mismatched) + 19 bp spacer	optimized (Dang et al. 2015)	7	Zeocin

Supplementary Table 3 | Comparison of PE2 (pegRNA) prediction tools. Overview table of currently available PE2 (pegRNA) prediction tools.

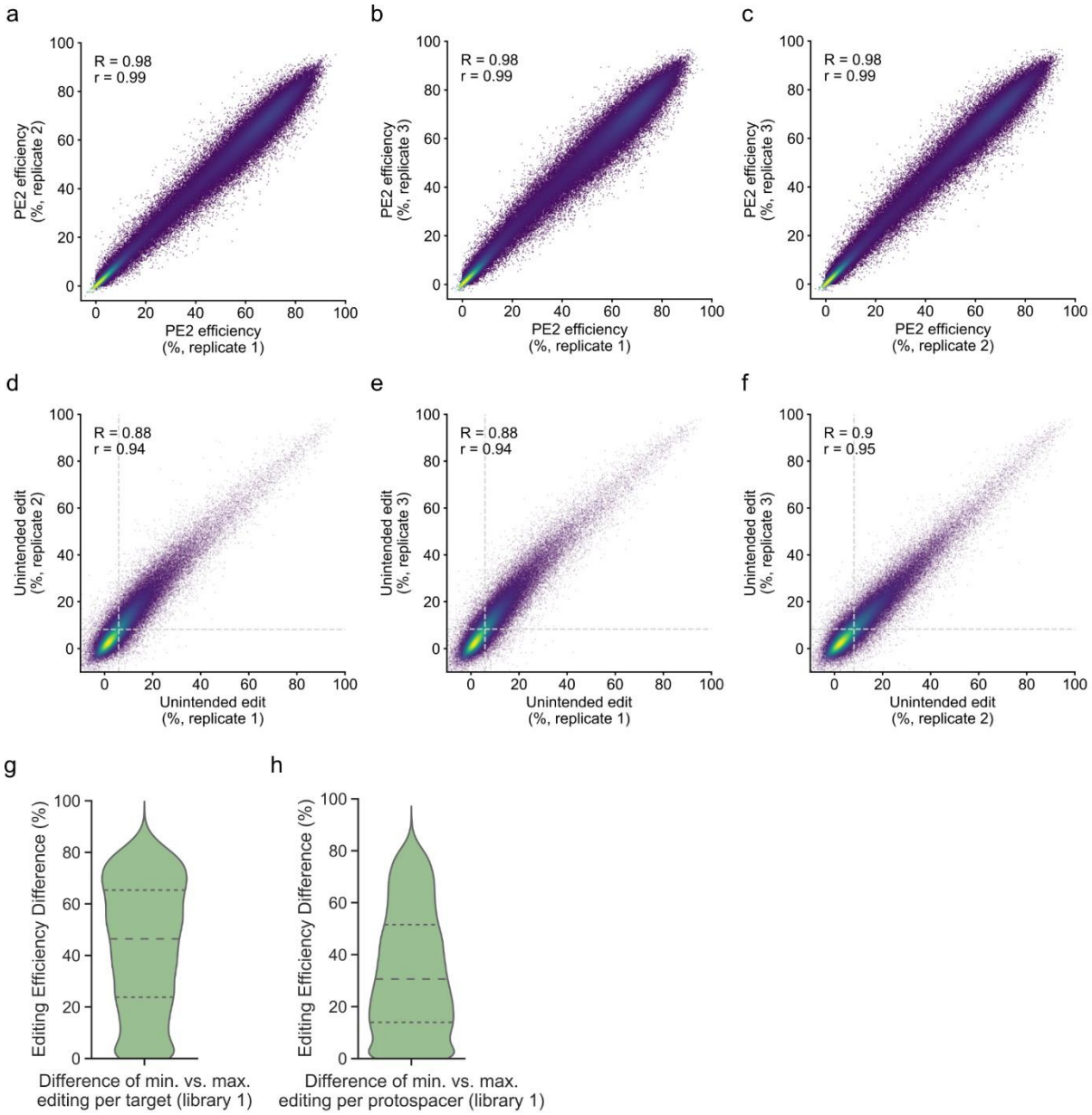
Model	Origin	Training dataset	Performance on test-dataset	Online tool	Restrictions in prediction edit type	Input format	Additional notes
PRIDICT	This study (Mathis et al.)	73,938 (diverse edits; replacements, insertions, deletions)	R = 0.85, r = 0.86	pridict.it	-	Target sequence with desired edit	Prediction of unintended editing rates, design of sequences for cloning, suggestions for nicking guide and NGS primers, option to select for pathogenic loci used in the library (+ visualizing library editing rates)
DeepPE	Kim et al.[4]	38,692 (G-to-C Pos5)	R = 0.8, r = 0.75	deepcrispr.info/DeepPE	Only G-to-C edits at position 5 of RTT	Target sequence only	
Position	Kim et al.[4]	1,774 (1-bp substitutions)	R = 0.56, r = 0.56		Only 1-bp substitutions (A-to-T, C-to-G, G-to-C, and T-to-A) at RTT positions 1-9 + 11 and 14	Target sequence only	
Type	Kim et al.[4]	3,775 (1-, 2-, 5-, 10-nt insertion, deletion, and substitution at position 1; two-position transversions at certain positions)	R = 0.47, r = 0.48		Four types of deletions, seven types of insertions, specific substitutions	Target sequence only	
Easy-Prime PE2 model	Li et al.[5]	41,898 (Kim et al.; DeepPE (87%), Position + Type datasets) + 199 (diverse edits from Anzalone et al.)	R = 0.67, r = 0.63 on full test-dataset; R = 0.54, r = 0.56 without 1bp edits at position 5	PE2-Model on GitHub (github.com/YichaoOU/easy_prime)	-	Input dataframe containing information about target and desired edit	

Supplementary Table 4 | Library sequences. List of library sequences used in this study, stored in separate file.

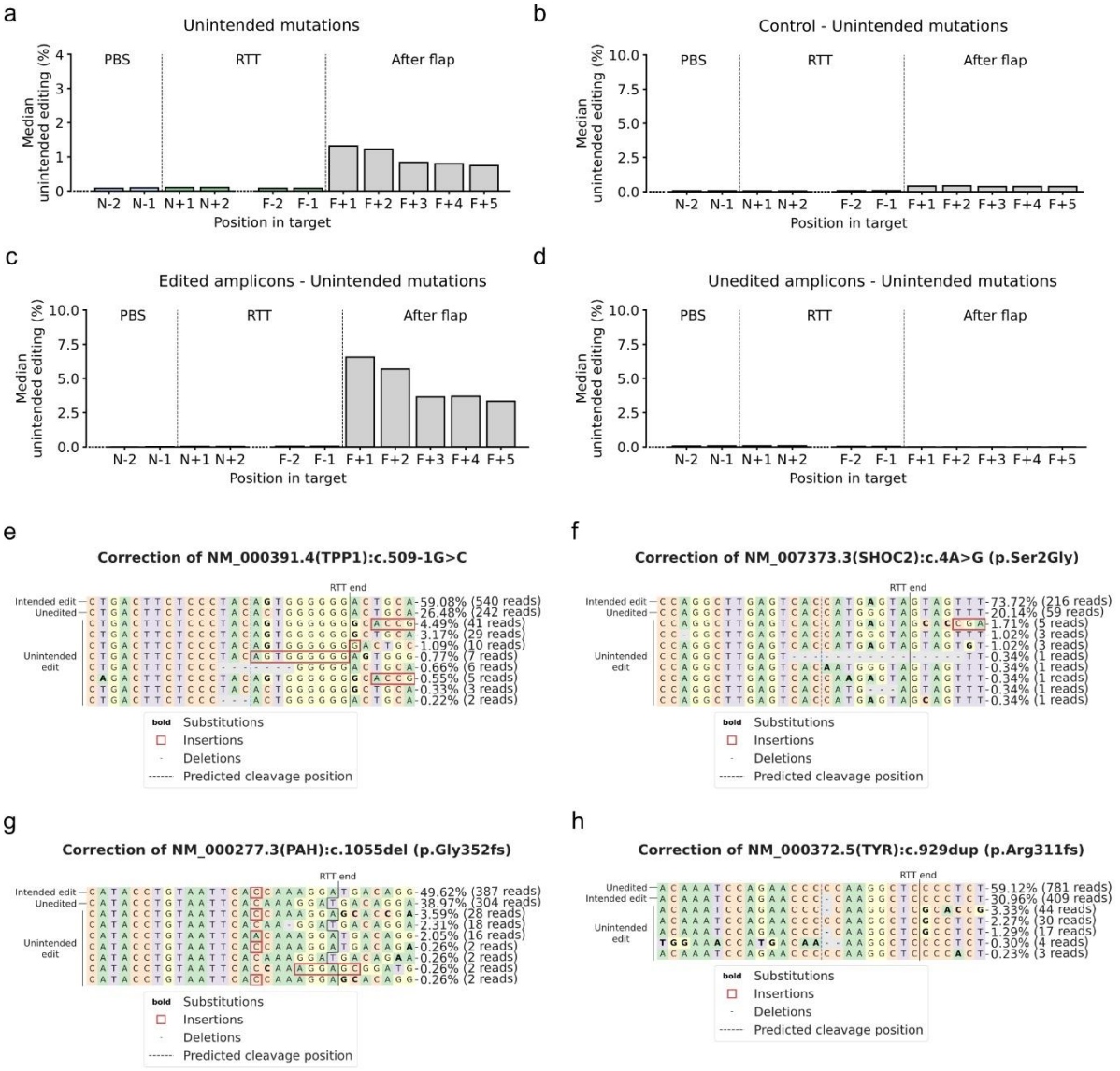
Supplementary Table 5 | Oligo sequences. List of oligo sequences used for cloning and PCR, stored in separate file.

Supplementary Table 6 | Editing rates of library 1, library 2 and endogenous editing experiments. Table containing information about pegRNAs used in library 1, library 2 and endogenous editing experiments and their associated editing efficiencies. Stored in separate file.

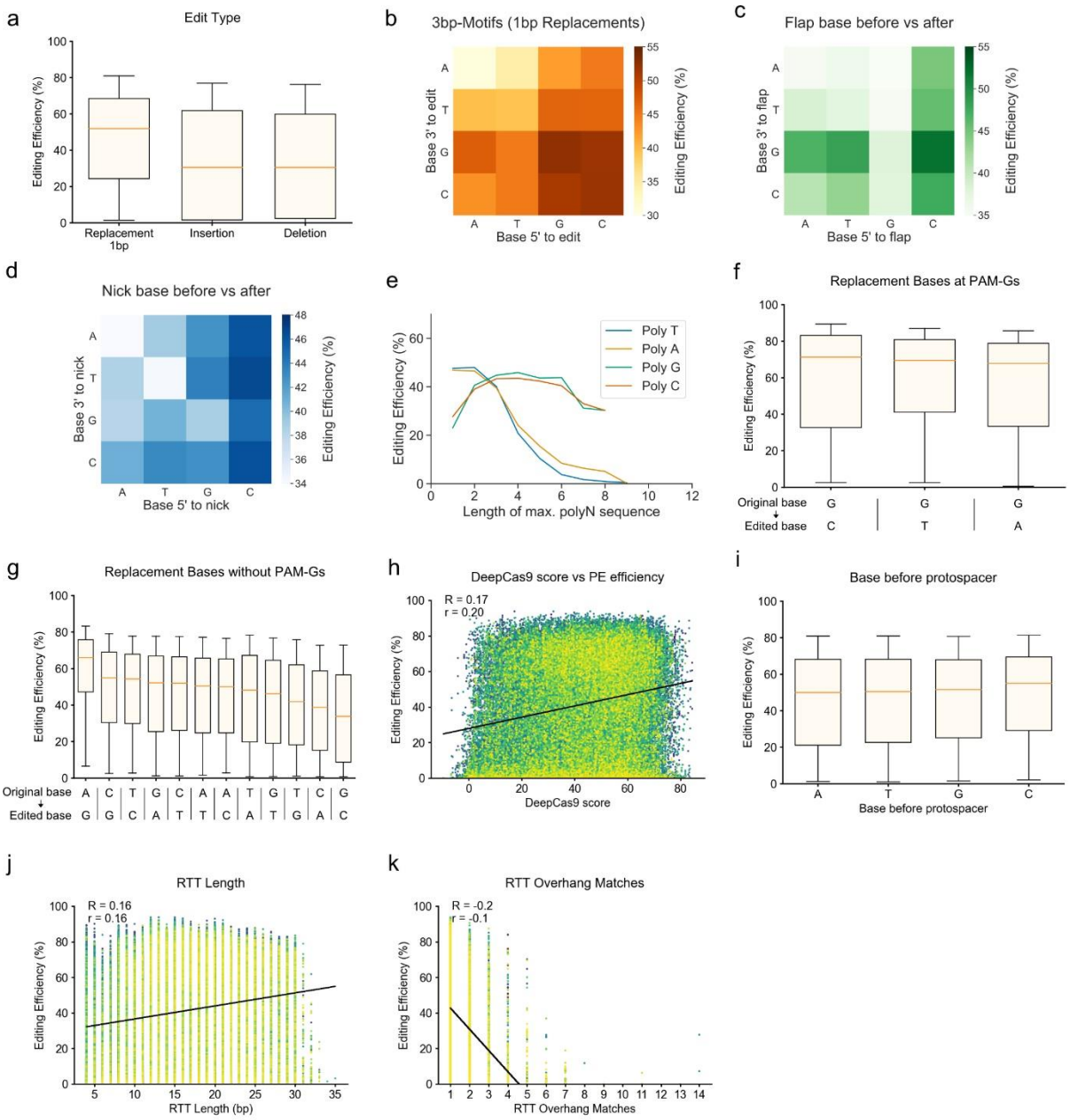
Supplementary Table 7 | Feature correlations of library 1. Table containing Spearman correlations between every pair of features listed in **Supplementary Table 1**. Stored in separate file.



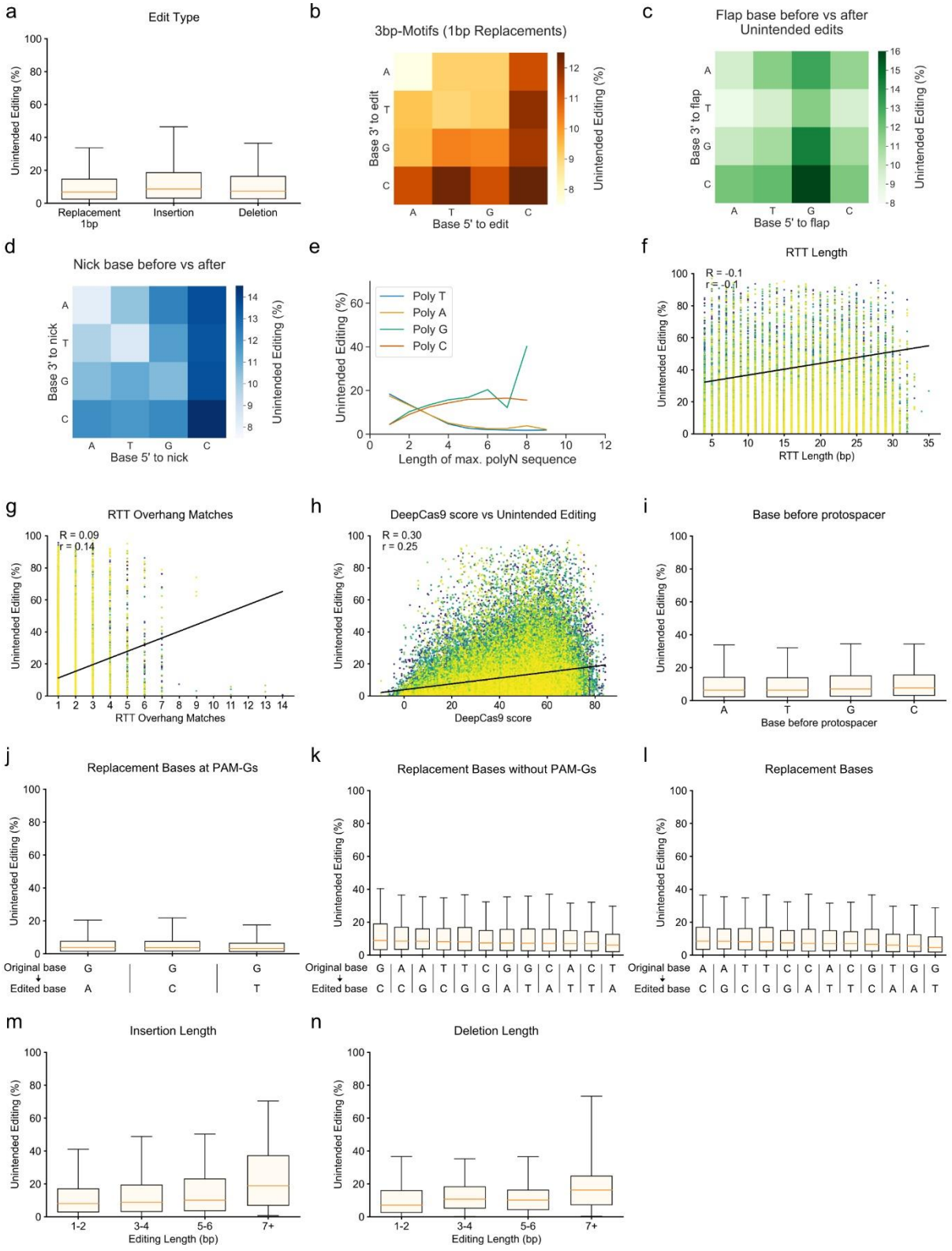
Supplementary Fig. 1 | Self-targeting screen replicate comparison and editing variability. (**a**) Correlation of editing between self-targeting screen (library 1) replicate 1 and 2. (**b**) Correlation of editing between self-targeting screen (library 1) replicate 1 and 3. (**c**) Correlation of editing between self-targeting screen (library 1) replicate 2 and 3. (**d**) Correlation of unintended editing between self-targeting screen (library 1) replicate 1 and 2. (**e**) Correlation of unintended editing between self-targeting screen (library 1) replicate 1 and 3. (**f**) Correlation of unintended editing between self-targeting screen (library 1) replicate 2 and 3. (**d-f**) The grey dashed line indicates the median unintended-editing rate. (**g**) Difference/Variation of minimum vs. maximum editing efficiency observed for different pegRNAs per target in library 1. (**h**) Difference/Variation of minimum vs. maximum editing efficiency observed for different pegRNAs per protospacer in library 1. (**a-h**) n = 92,423.



Supplementary Fig. 2 | Characterization of unintended editing in library 1. (**a-d**) Percentage of sequence mutations at certain base positions in PBS, RTT and region after flap. Positions around nick (N) and flap (F) are presented. (**a**) Overall unintended editing (edited and unedited amplicons) in samples with PE2. (**b**) Unintended editing in the control condition. Low values of unintended editing can be observed after the flap, which is caused by recombining the extension template with the target during PCR. Control unintended editing values were subtracted in **a**, **c** and **d** as well as in the unintended editing rates of library 1 (described in the methods section) to reduce the influence of PCR recombination on the final model. (**c**) Edited amplicons in samples with PE2. (**d**) Unedited amplicons in samples with PE2. (**e-h**) Visualizations of read distribution of 4 different pegRNAs in library 1. (**e**) Replacement (C to G). (**f**) Replacement (G to A). (**g**) Insertion (+C) (**h**) Deletion (-C).

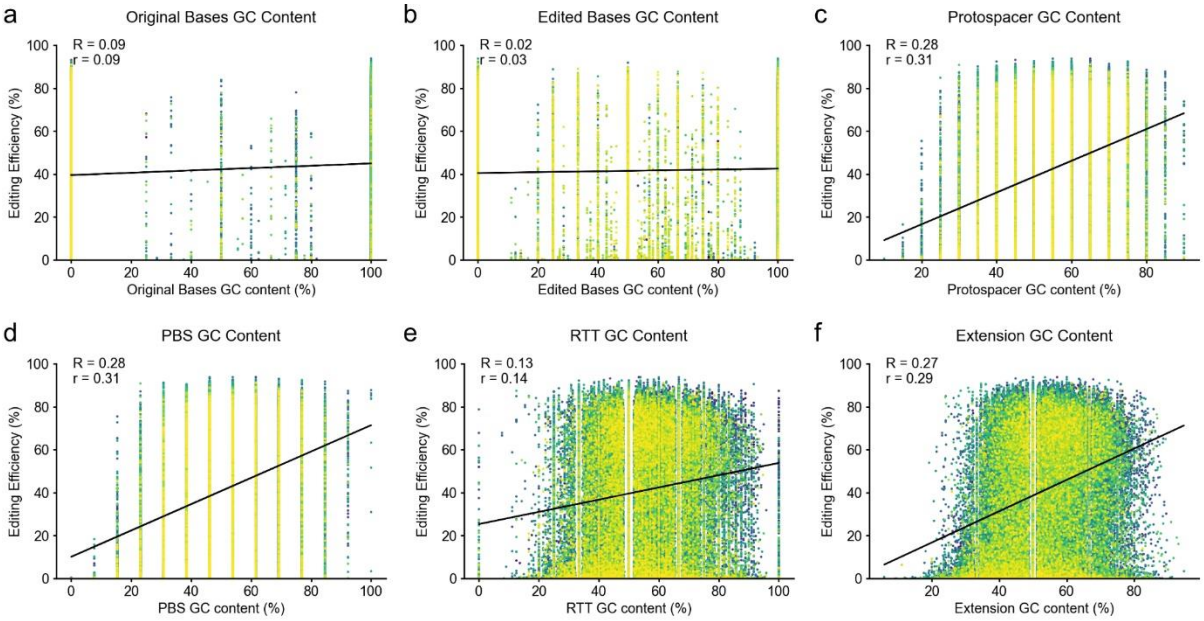


Supplementary Fig. 3 | Visualization of intended editing features used in training statistical ML models. (**a**) Single base replacement- (n = 57,920) vs. insertion- (n = 57,920) vs. deletion-edits (n = 6,083). (**b**) Base 5' and 3' of single base replacement edit. (**c**) Base 5' to edited flap end and base 3' to edited flap end. (**d**) Base 5' to nick position and base 3' to nick position (= edit position 1). (**e**) Length of the maximum number of identical sequential bases (polyA/polyT/polyG/polyC) in spacer and extension sequence. (**f**) Replacement base if edited at PAM-Gs. n = 1,751, 3,173, 2,301. (**g**) Replacement base if edited at all positions except PAM-Gs. n = 4,482, 3,622, 5,252, 3,591, 3,768, 4,868, 4,644, 4,864, 3,403, 4,832, 3,353, 4,016. (**h**) DeepSpCas9 score (Kim et al. 2019) of pegRNA. (**i**) Base before/upstream of protospacer (5'). n = 16,694, 13,792, 12,861, 14,573. (**j**) RTT length. (**k**) The number of times the RTT overhang sequence matches the target sequence within 15 bases downstream of the edit position. (**h,j,k**) The black line corresponds to the least-squares polynomial fit. Boxplots in (**a**,**f**,**g**,**i**) represent the 25th, 50th, and 75th percentiles. Whiskers indicate 5 and 95 percentiles.

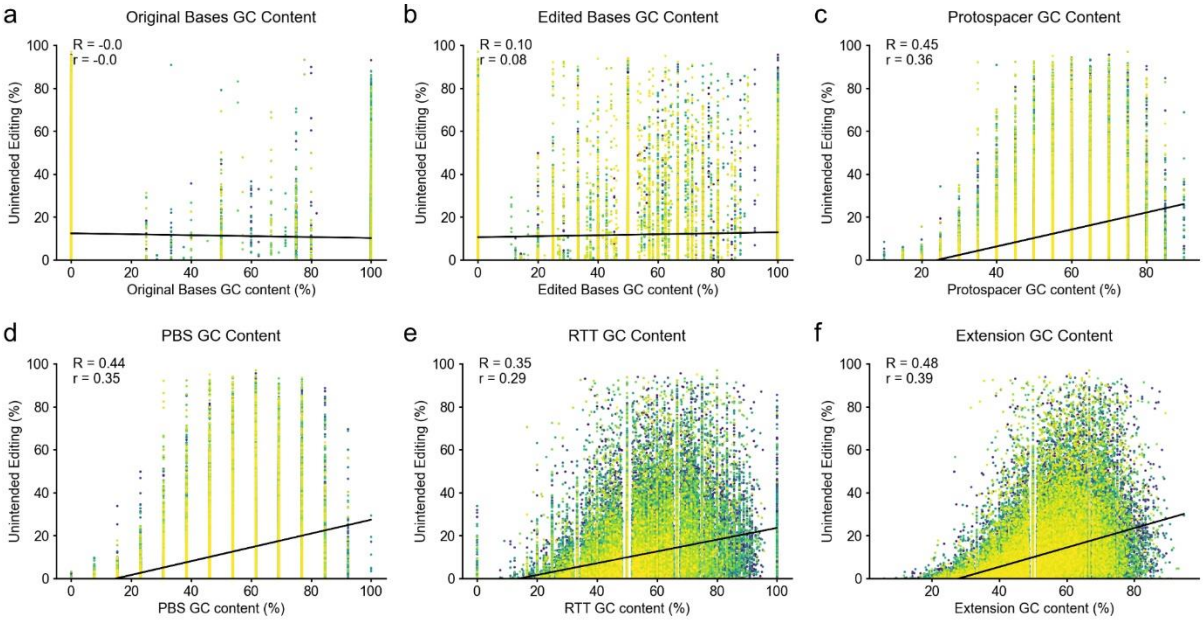


Supplementary Fig. 4 | Visualization of unintended editing features used in training statistical ML models. (**a**) Single base replacement- (n = 57,920) vs. insertion- (n = 57,920) vs. deletion-edits (n = 6,083). (**b**) Base 5' and 3' of single base replacement edit. (**c**) Base 5' to edited flap end and base 3' to edited flap end. (**d**) Base 5' to nick position and base 3' to nick position (= edit position 1). (**e**) Length of the maximum number of identical sequential bases (polyA/polyT/polyG/polyC) in spacer and extension sequence. (**f**) RTT length. (**g**) The number of RTT overhang sequence matches to the target

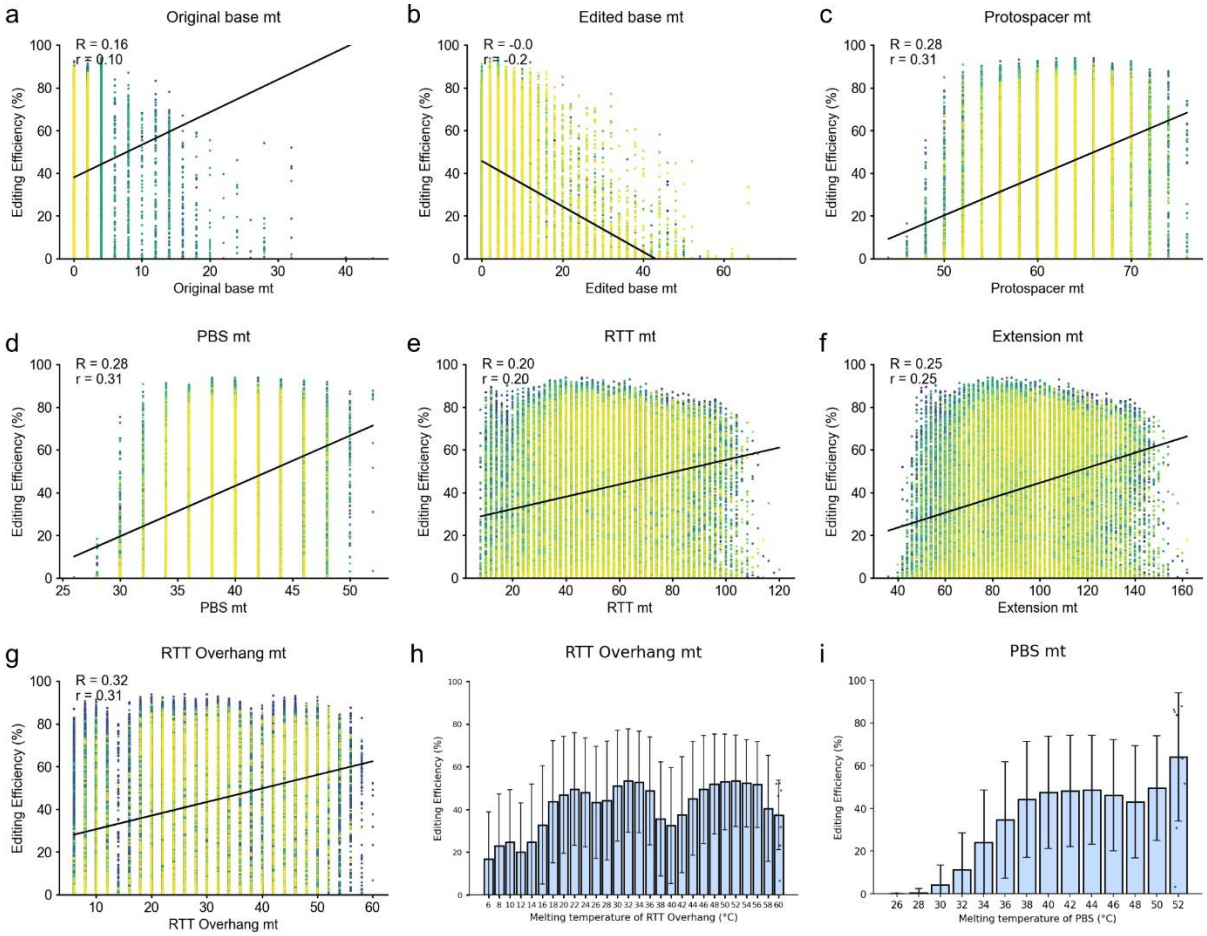
sequence within 15 bases downstream of the edit position. (**h**) DeepSpCas9 score (Kim et al. 2019) of pegRNA. (**f-h**) The black line corresponds to the least-squares polynomial fit. (**i**) Base upstream of protospacer (5'). n = 16,694, 13,792, 12,861, 14,573. (**j**) Replacement base if edited at PAM-Gs. n = 2,301, 1,751, 3,173. (**k**) Replacement base if edited at all positions except PAM-Gs. n = 4,016, 4,644, 4,482, 5,252, 4,832, 3,622, 3,591, 3,403, 3,353, 4,868, 3,768, 4,864. (**l**) Replacement base if edited at any position. n = 4,644, 4,482, 5,252, 4,832, 3,622, 3,353, 4,868, 3,768, 5,767, 4,864, 5,892, 6,576. (**m**) Insertion length. n = 21,882, 2,934, 1,103, 2,501. (**n**) Deletion length. n = 5,457, 459, 115, 52. Boxplots in (**a,i-n**) represent the 25th, 50th, and 75th percentiles. Whiskers indicate 5 and 95 percentiles.



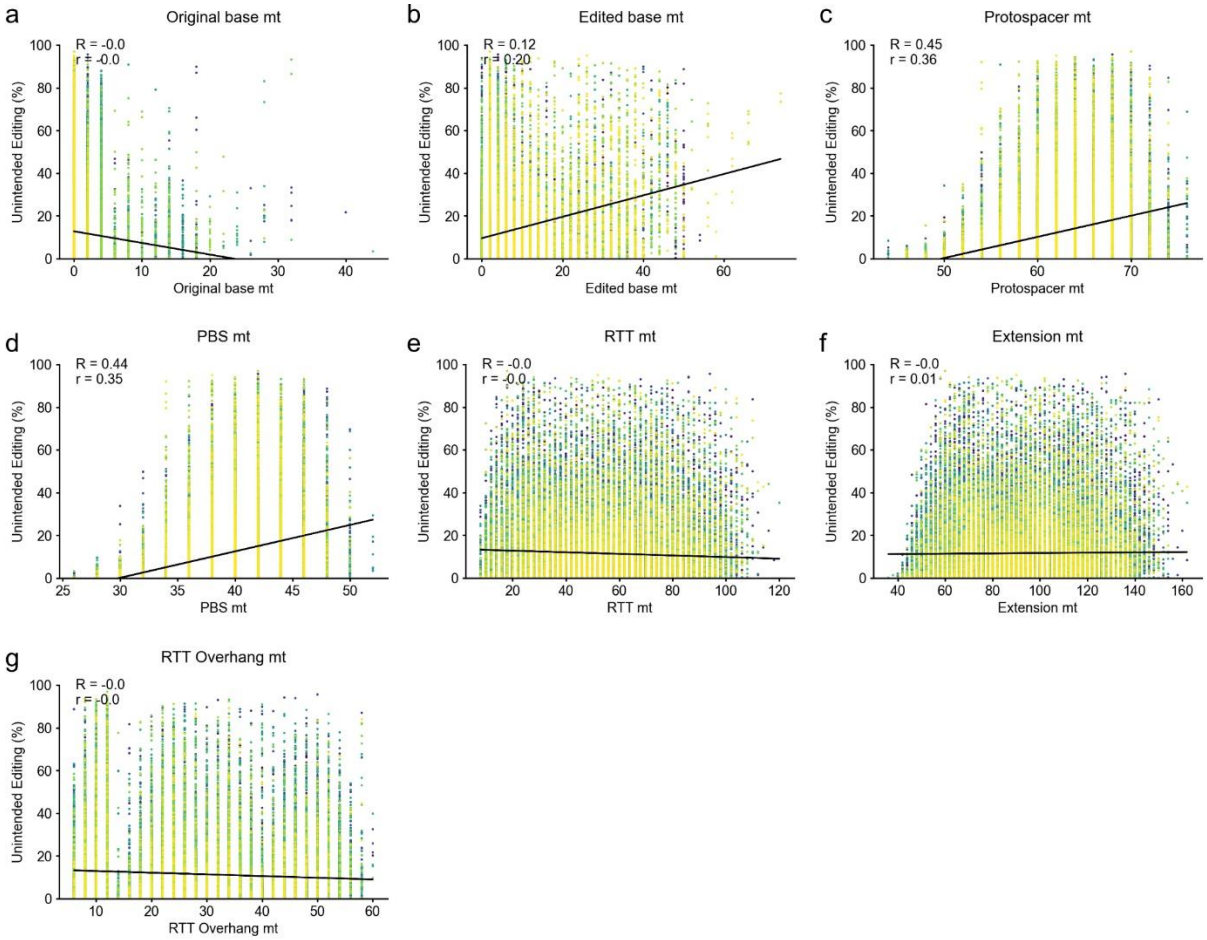
Supplementary Fig. 5 | Visualization of GC content effect on editing efficiency. (**a**) GC content of original edit base/sequence stretch. (**b**) GC content of edited base/sequence stretch. (**c**) GC content in protospacer. (**d**) GC content in PBS. (**e**) GC content in RTT. (**f**) GC content in total extension (PBS+RTT). (**a-f**) Black line corresponds to least squares polynomial fit. n = 92,423.



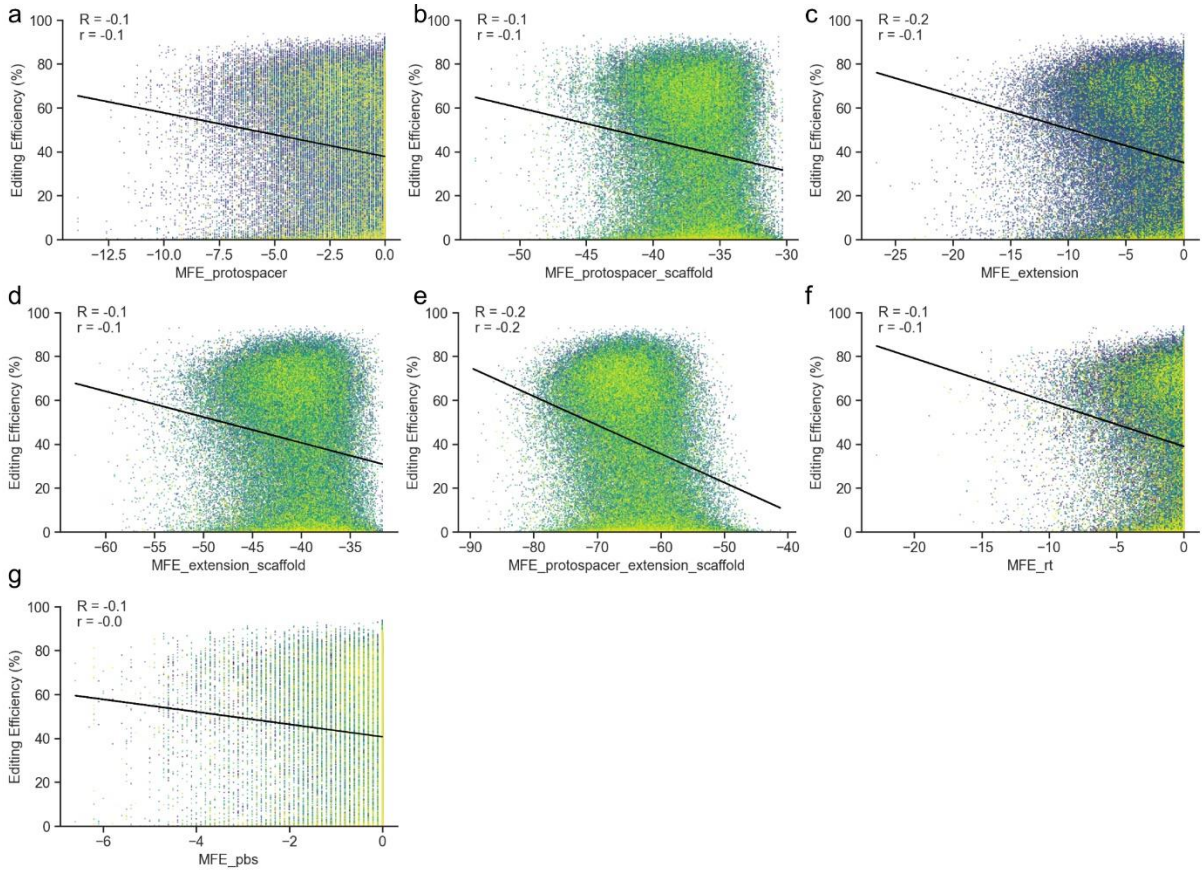
Supplementary Fig. 6 | Visualization of GC content effect on unintended editing efficiency. (**a**) GC content of original edit base/sequence stretch. (**b**) GC content of edited base/sequence stretch. (**c**) GC content in protospacer. (**d**) GC content in PBS. (**e**) GC content in RTT. (**f**) GC content in total extension (PBS+RTT). (**a-f**) Black line corresponds to least squares polynomial fit. n = 92,423.



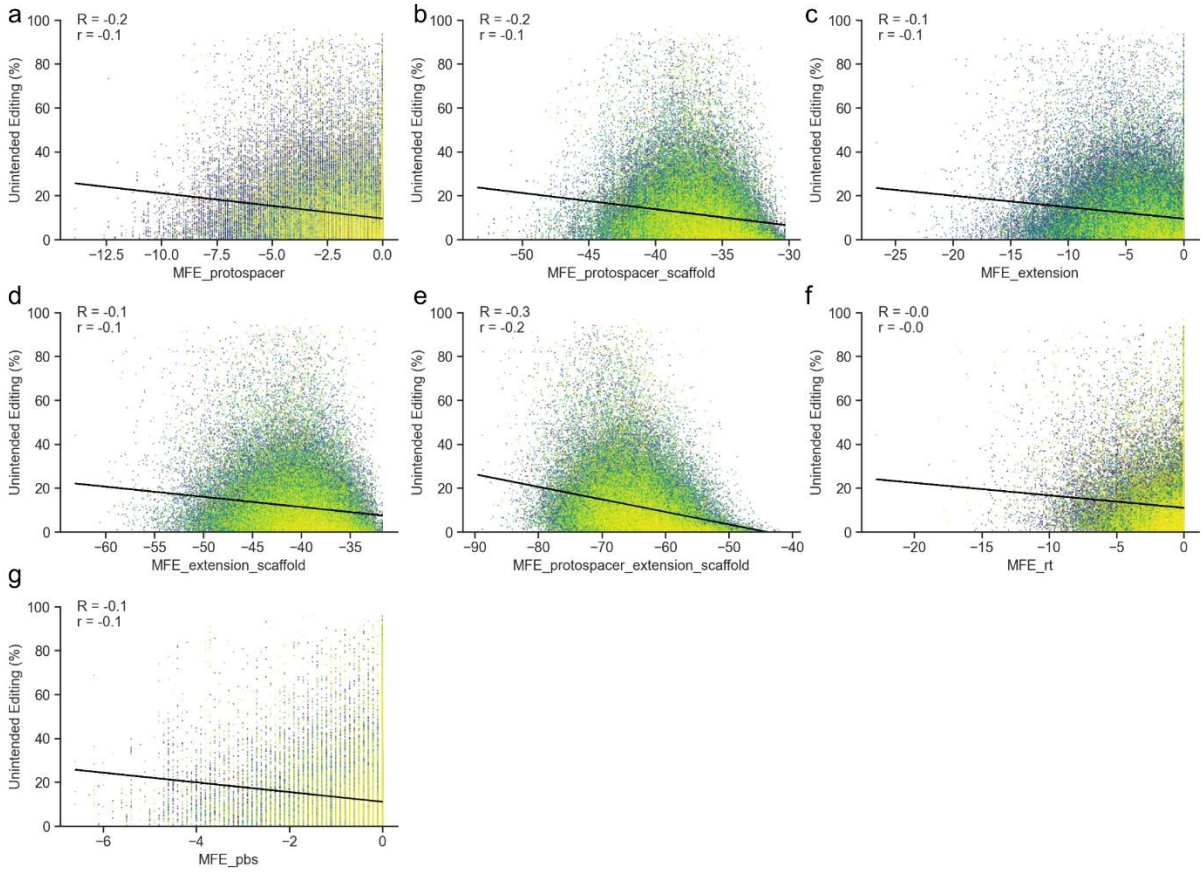
Supplementary Fig. 7 | Visualization of melting temperature effect on editing efficiency. (**a**) Melting temperature of original edit base/sequence stretch. (**b**) Melting temperature of base/sequence stretch after edit. (**c**) Melting temperature of protospacer. (**d**) Melting temperature of PBS. (**e**) Melting temperature of RTT. (**f**) Melting temperature of full extension (PBS+RTT). (**g**) Melting temperature of RTT overhang sequence. (**h**) Barplot visualization of RTT overhang melting temperature. n for each bar (from left to right) = 1,726, 6,825, 10,705, 4,412, 127, 695, 2,347, 5,251, 7,164, 5,964, 3,943, 3,868, 5,043, 5,441, 4,151, 2,127, 1,277, 1,460, 2,448, 3,468, 4,015, 3,968, 3,021, 1,858, 778, 261, 72, 8. (**i**) Barplot visualization of PBS melting temperature. n for each bar (from left to right) = 23, 305, 1,349, 3,293, 7,120, 12,336, 17,562, 19,303, 16,058, 10,293, 3,796, 865, 111, 9. (**h,i**) Individual data points shown for bars with n < 30. Error bars = mean +/- SD (**a-g**) The black line corresponds to the least-squares polynomial fit. n = 92,423.



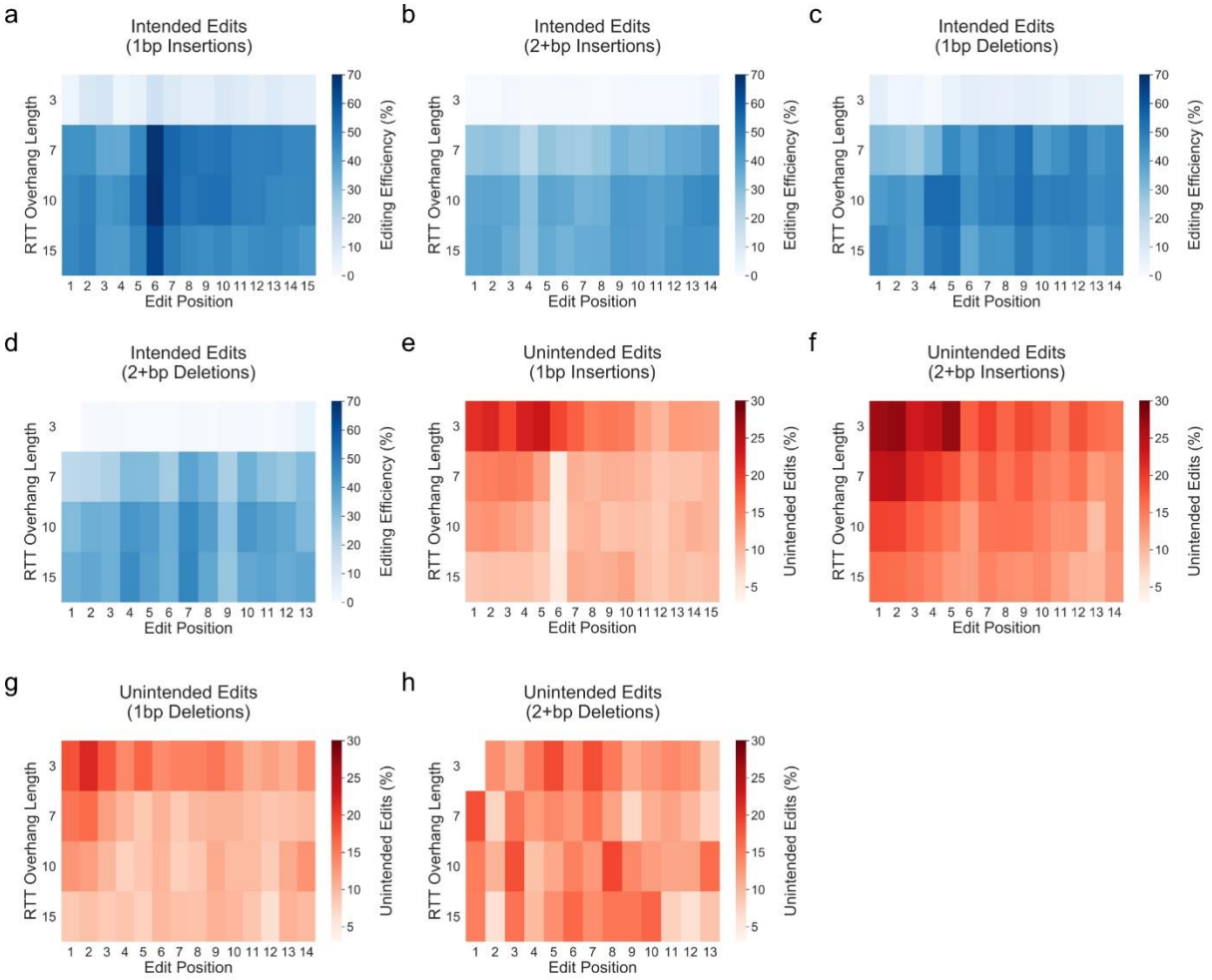
Supplementary Fig. 8 | Visualization of melting temperature effect on unintended editing efficiency. (**a**) Melting temperature of original edit base/sequence stretch. (**b**) Melting temperature of base/sequence stretch after edit. (**c**) Melting temperature of protospacer. (**d**) Melting temperature of PBS. (**e**) Melting temperature of RTT. (**f**) Melting temperature of full extension (PBS+RTT). (**g**) Melting temperature of RTT overhang sequence. (**a-g**) The black line corresponds to the least-squares polynomial fit. n = 92,423.



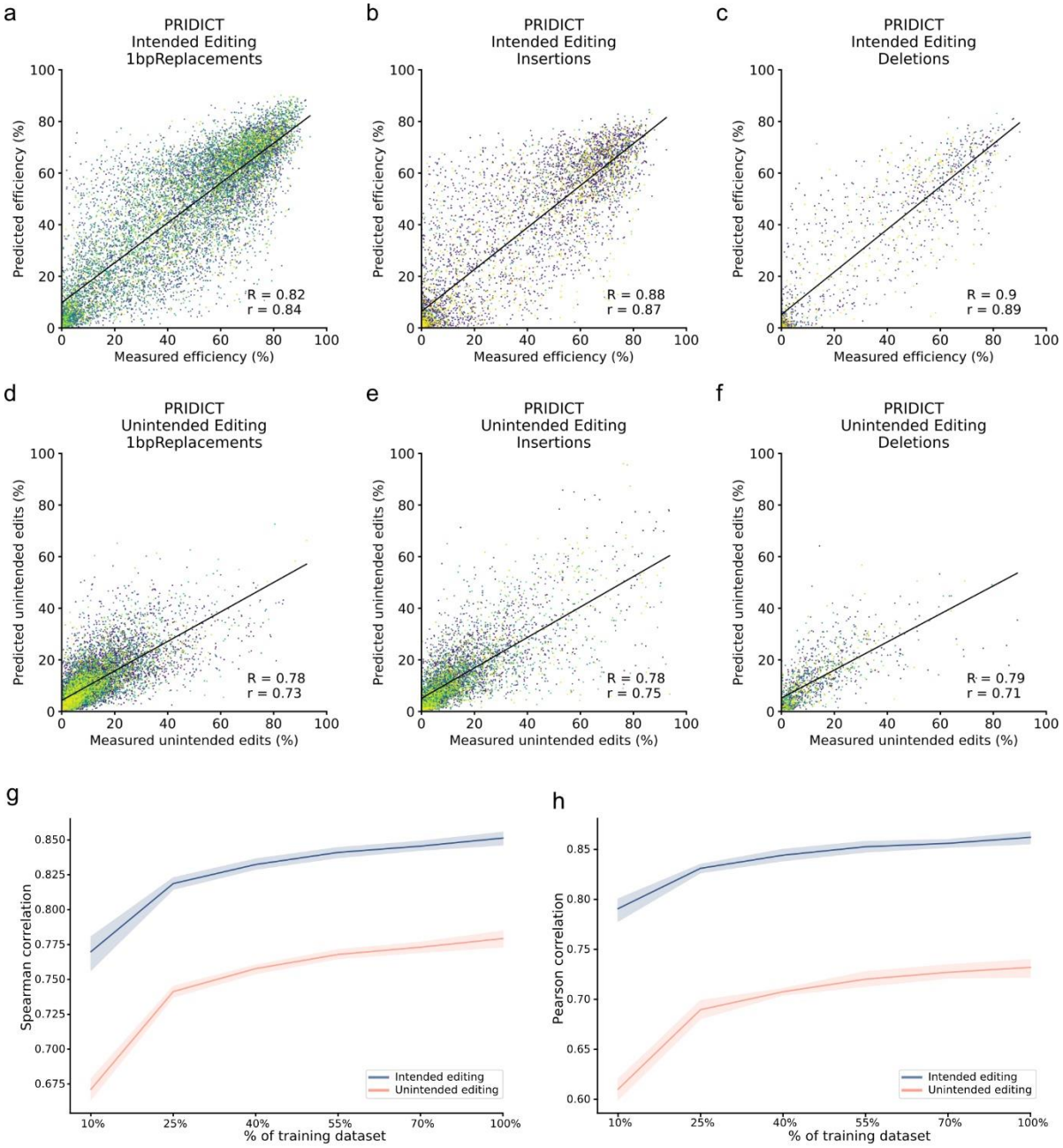
Supplementary Fig. 9 | Visualization of the effect of minimum free energy (MFE) on editing efficiency. (**a**) MFE of spacer sequence. (**b**) MFE of spacer sequence with the scaffold. (**c**) MFE of extension sequence. (**d**) MFE of the scaffold with extension. (**e**) MFE of full pegRNA (spacer with scaffold and extension. (**f**) MFE of RTT. (**g**) MFE of PBS. (**a-g**) Lower MFE corresponds to more structured/stable RNA folding. The black line corresponds to the least-squares polynomial fit. n = 92,423.



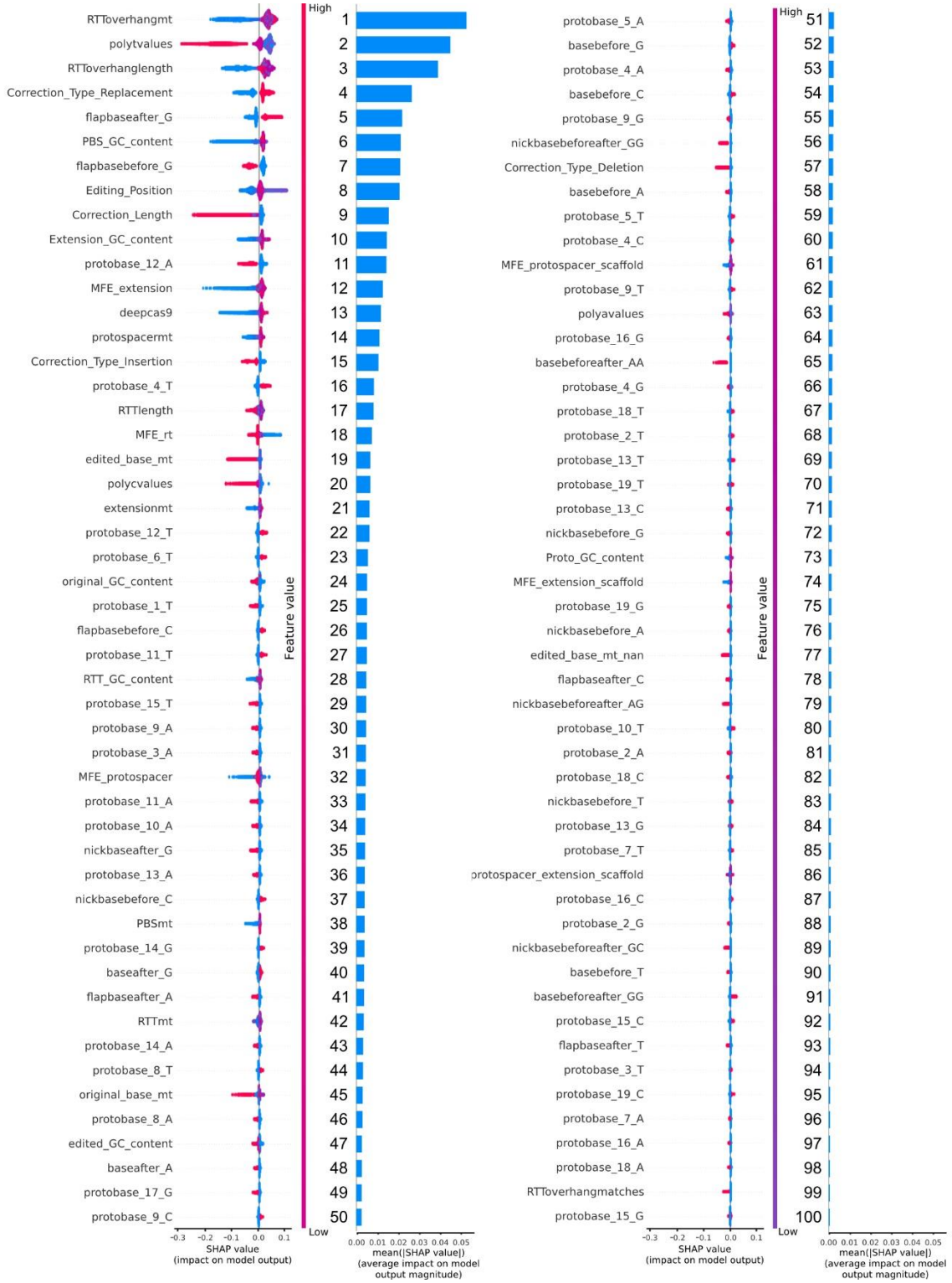
Supplementary Fig. 10 | Visualization of the effect of minimum free energy (MFE) on unintended editing efficiency. (**a**) MFE of spacer sequence. (**b**) MFE of spacer sequence with the scaffold. (**c**) MFE of extension sequence. (**d**) MFE of the scaffold with extension. (**e**) MFE of full pegRNA (spacer with scaffold and extension. (**f**) MFE of RTT. (**g**) MFE of PBS. (**a-g**) Lower MFE corresponds to more structured/stable RNA folding. The black line corresponds to the least-squares polynomial fit. n = 92,423.



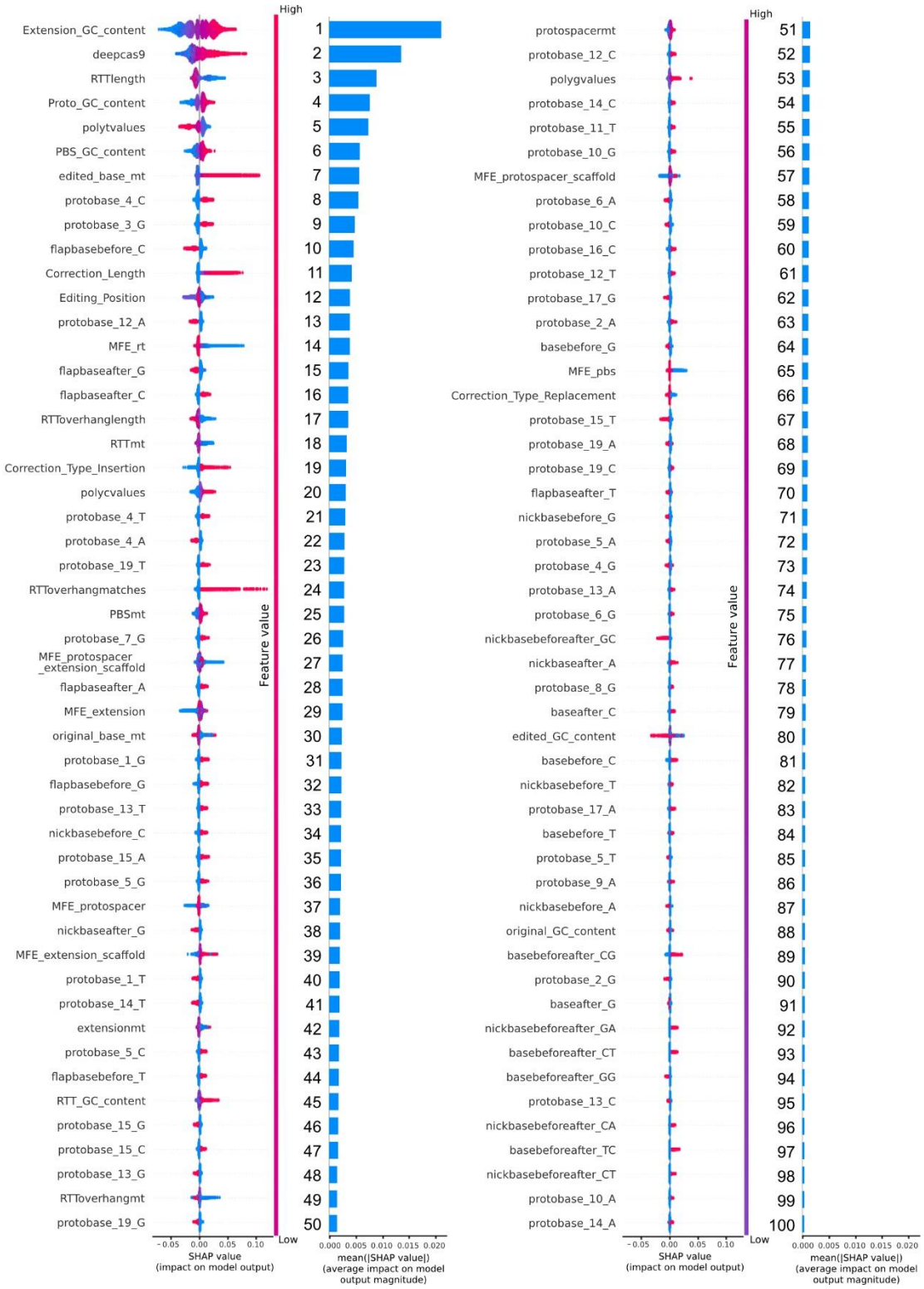
Supplementary Fig. 11 | Effect of RTT overhang length and edit position on editing efficiency and unspecific edits. (**a-d**) Mean editing efficiency of different editing types with edit position and RTT overhang length. (**a**) 1bp insertions. (**b**) 2bp or longer insertions. (**c**) 1bp deletions. (**d**) 2bp or longer deletions. (**e-h**) Mean rate of unintended edits of different editing types with edit position and RTT overhang length. (**e**) 1bp insertions. (**f**) 2bp or longer insertions. (**g**) 1bp deletions. (**h**) 2bp or longer deletions. (**h**) 2bp or longer deletions.



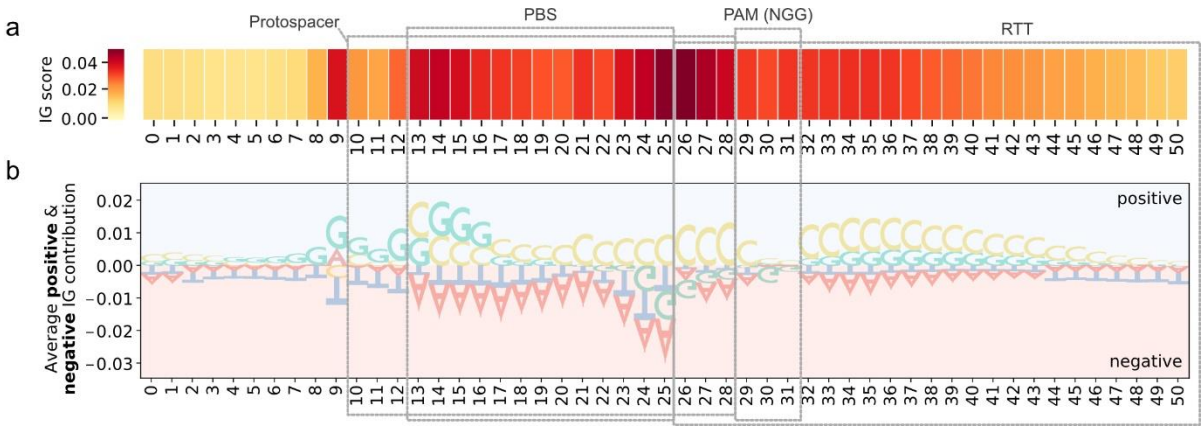
Supplementary Fig. 12 | Performance of PRIDICT model on different edit types. (**a-c**) Visualization of predicted and measured editing efficiency for (**a**) Single base replacement edits, (**b**) insertions, and (**c**) deletions. (**d-f**) Visualization of predicted and measured unintended editing rates for (**d**) single base replacement edits, (**e**) insertions, and (**f**) deletions. (**a-f**) Predictions were taken from run 2 of five-fold cross-validation. (**g**) Performance of PRIDICT (Spearman correlation) when training on fractions of the training dataset. (**h**) Performance of PRIDICT (Pearson correlation) when training on fractions of the training dataset. (**g,h**) Shaded area corresponds to the mean +/- SD of 5-fold cross-validation.



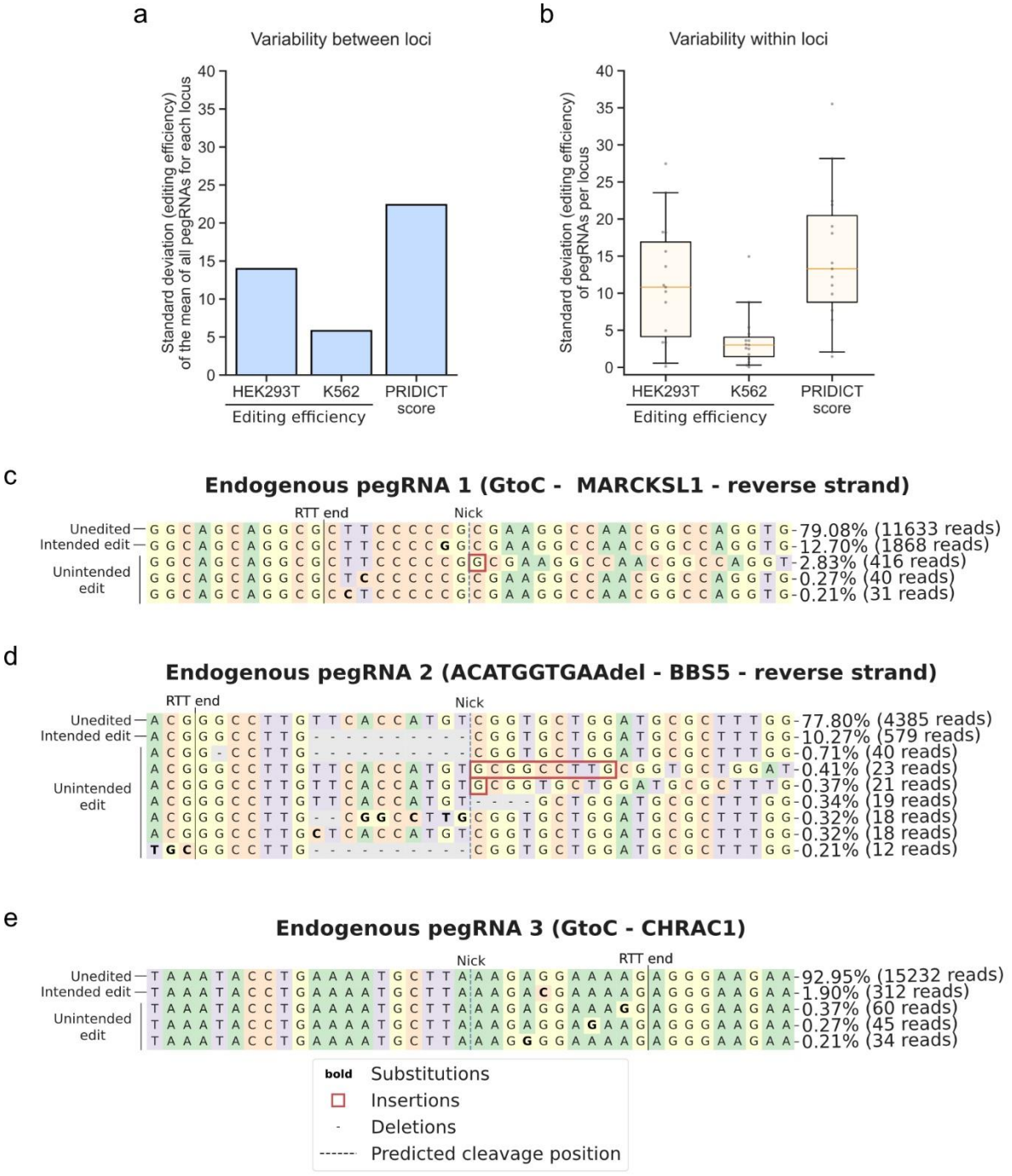
Supplementary Fig. 13 | Intended Editing - SHAP Analysis of XGBoost model trained on the self-targeting dataset – Top 100. SHAP analysis of XGBoost model (intended editing) visualizing top 100 features. Dot plots: High SHAP value associates with higher editing prediction. Feature values correspond to the values of each individual feature. Barplots: mean ([SHAP value]) corresponds to average impact on model output magnitude. Features with higher values provide a higher impact on model output.



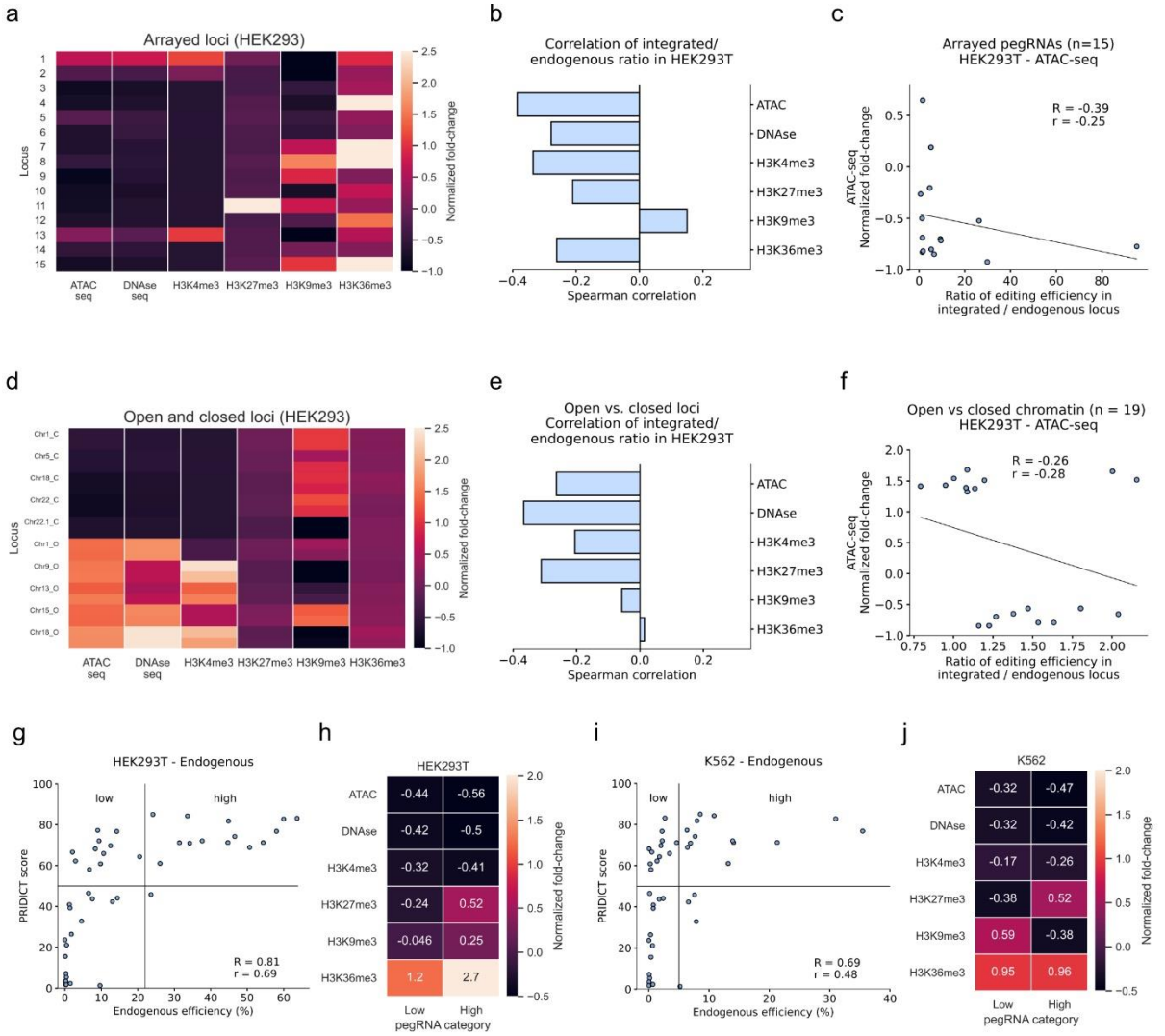
Supplementary Fig. 14 | Unintended Editing - Full SHAP Analysis of XGBoost model trained on the self-targeting dataset – Top 100. SHAP analysis of XGBoost model (unintended editing) visualizing top 100 features. Dot plots: High SHAP value associates with a higher unintended editing prediction. Feature values correspond to the values of each individual feature. Barplots: mean([SHAP value]) corresponds to average impact on model output magnitude. Features with higher values provide a higher impact on model output.



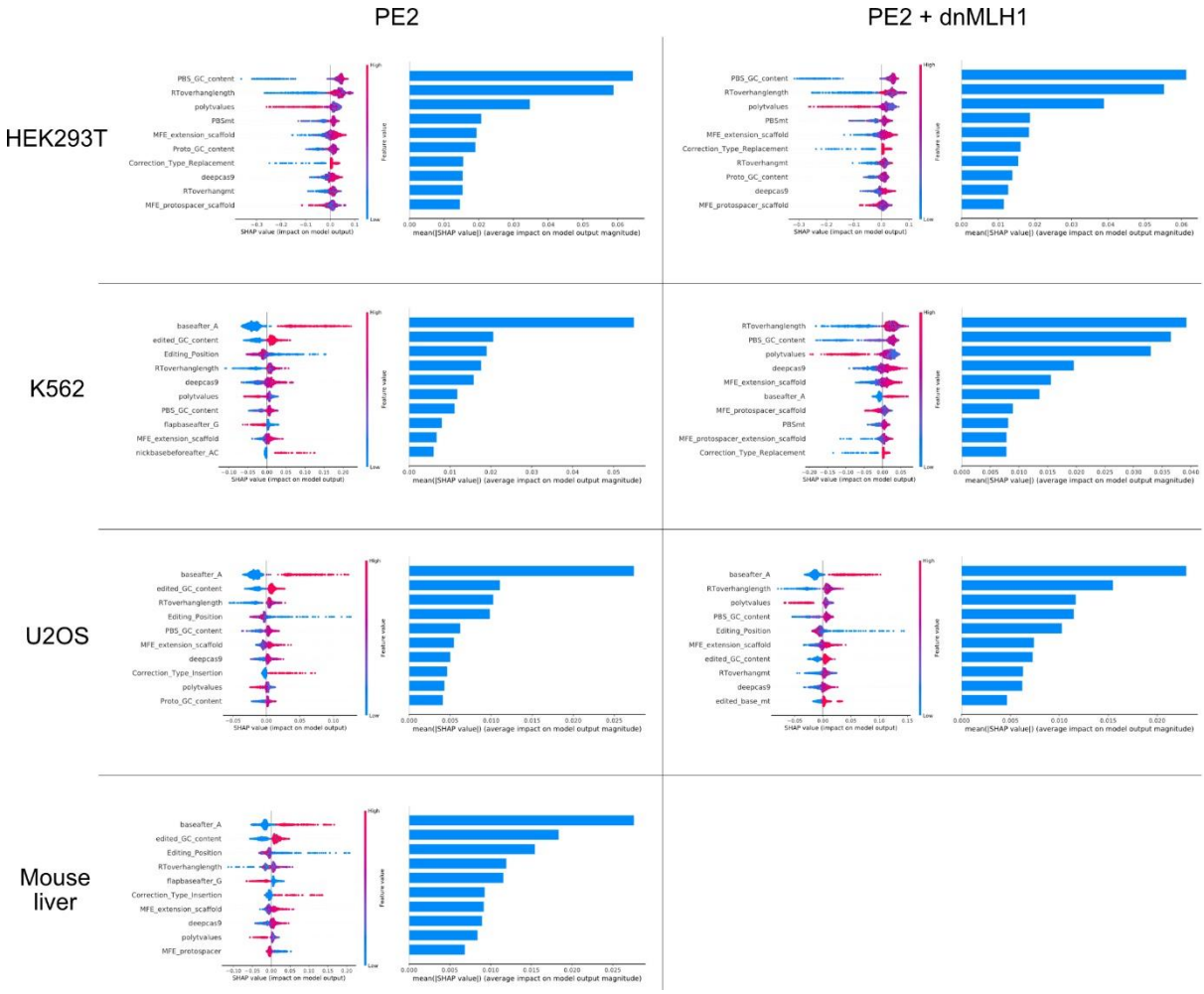
Supplementary Fig. 15 | Feature importance analysis with integrated gradient for unintended editing prediction. (**a**) Analysis of the importance of individual sequence positions in the PRIDICT AttnBiRNN model. A high IG score associates with a high impact on editing prediction. (**b**) Average positive/negative IG contribution of individual positions. Large bases have a stronger effect on prediction.



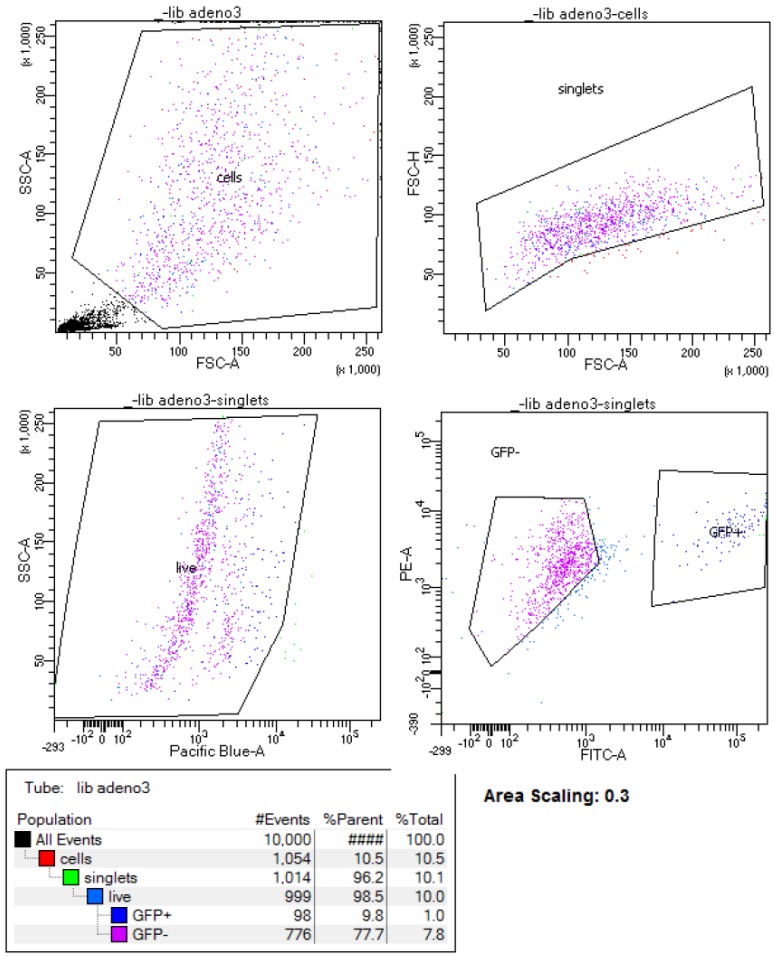
Supplementary Fig. 16 | Additional analysis of endogenous editing experiments. (**a**) Bars show variability between 15 loci by showing the standard deviation of the mean editing efficiency of 3 pegRNAs for each locus. (**b**) Variability of editing efficiency of 3 endogenous pegRNAs within each locus. Each grey dot corresponds to the standard deviation of pegRNAs for a given locus (15 loci). n = 15 for each boxplot. Boxplots represent the 25th, 50th, and 75th percentiles. Whiskers indicate 5 and 95 percentiles. (**c**-**e**) Visualizations of read distribution of 3 different pegRNAs targeting endogenous loci. (**c**) GtoC mutation in MARCKSL1 gene. (**d**) ACATGGTGAA deletion in BBS5 gene. (**e**) GtoC mutation in CHRAC1 gene.



Supplementary Fig. 17 | Analysis of the effect of chromatin characteristics on prime editing. (**a**) Levels of different chromatin characteristics from publicly available datasets of HEK293 (ATAC-seq, DNAse-seq, H3K4me3, H3K27me3, H3K9me3, and H3K36me3) at the 15 loci targeted in **Fig. 4a**. Values are normalized across analyzed loci in **a** and **d**. (**b**) Correlation between the ratio of intended editing at integrated loci (library screen)/endogenous loci vs. chromatin characteristics. (**c**) Visualization of ATAC-seq score vs. integrated/endogenous intended editing efficiency ratio of arrayed pegRNAs. (n = 15) (**d**) Levels of different chromatin characteristics from publicly available datasets of HEK293 (ATAC-seq, DNAse-seq, H3K4me3, H3K27me3, H3K9me3, and H3K36me3) at 10 loci (2 pegRNAs each) that were previously targeted by Cas9 and classified as open (n = 5) or closed (n = 5) regions[25]. Values are normalized across analyzed loci in **a** and **d**. (**e**) Correlation of the integrated/endogenous intended editing ratio vs. chromatin characteristics with pegRNAs targeting closed (ATAC-seq < -0.5; n = 9) or open regions (ATAC-seq > 1.3; n = 10). (**f**) Visualization of ATAC-seq score vs. integrated/endogenous intended editing efficiency ratio of pegRNAs targeting closed (ATAC-seq < -0.5; n = 9) or open regions (ATAC-seq > 1.3; n = 10). (**g**) Distinguishing pegRNAs with high PRIDICT score but low or high endogenous efficiency in HEK293T. Datapoints from **Fig. 4a** (n = 45). (**h**) Average levels of selected chromatin characteristics (in HEK293) of pegRNAs with high PRIDICT score and low (upper left corner) vs. high (upper right corner) editing rates in HEK293T. (**i**) Distinguishing pegRNAs with high PRIDICT score but low or high endogenous efficiency in K562. Datapoints from **Fig. 4b** (n = 45). (**j**) Average levels of selected chromatin characteristics of pegRNAs with high PRIDICT score and low (upper left corner) vs. high (upper right corner) editing rates in K562.



Supplementary Fig. 18 | SHAP analysis of XGBoost models trained on library 2 datasets. SHAP analysis of HEK293T, K562, U2OS, and mouse liver datasets from XGBoost models trained and tested on library 2 with tevopreQ1. Complete datasets (without splitting in train/test datasets) were used for training and testing since the trained XGBoost models are only trained for extracting feature importance in different experimental contexts via SHAP analysis. Descriptions of the different features are listed in **Supplementary Table 1**. Feature values correspond to the values of each individual feature. Barplots: mean([SHAP value]) corresponds to average impact on model output magnitude. Features with higher values provide a higher impact on model output.



Supplementary Fig. 19 | Gating strategy for library 2 FACS of mouse hepatocytes. FACS gating strategy for sorting GFP-positive mouse hepatocytes containing integrated library 2.

Supplementary Methods 1

Supplementary Methods 1a:

1.1 PRIDICT Model Overview

We designed and implemented **PRIDICT** model in PyTorch [1] comprising three encoders and one decoder neural network. Two separate encoder networks used bidirectional attention-based recurrent neural networks (RNN) to encode and learn representations from a pair of ***original*** and ***mutated*** sequences, and a third encoder network that used a feed-forward network to encode and learn a representation from a set of derived/computed ***sequence-level*** features (such as melting temperature, minimum free energy, etc.). These learned representations (i.e. fixed-length vectors) were mapped using a decoder network to learn a probability distribution on three outcomes associated with ***edited***, ***unedited***, and ***unintended edit*** proportions.

The model's two **sequence encoder** networks had similar architecture and comprised of the following blocks:
An (1) **Embedding block** that embeds the nucleotides (i.e. ACGT letters) and their corresponding position annotations (binary indicators such as being part of protospacer, PBS or RTT) from one-hot encoded representation to a dense vector representation.
An (2) **Encoder block** that contains a multilayer bidirectional gated recurrent units (GRU) [2, 3], that takes the embeded representations from the **Embedding block** and computes another representation based on left-to-right and right-to-left processing of the tokens in the sequence (i.e. uses context representation in both directions) when learning the new vector representation of the different tokens.
Lastly, an (3) **Attention** block comprising of (a) global context/query vector $\overline{q}_{global}$ (i.e. trainable parameters) that pools (using weighted averaging based on attention scores) the sequence of token-level representations to one representation (i.e. fixed-length vector), and (b) RTT query vector $\overline{q}_{RTT}$ to pool token-level representations belonging to the RTT (i.e. contiguous part of the sequence where mutation occurs) to a fixed-length vector representation.
For the ***sequence-level*** derived features, the model's encoder network is comprised of a mix of **multilayer feed-forward** networks, residual connections and layer-normalization operations to map the input features to a dense vector representation.
Lastly, the models' **decoder network** stacks the five computed representations from the three encoder networks and uses a series of **multilayer feed-forward** networks, residual connections and layer-normalization layers to compute a probability distribution vector on the three outcomes representing ***edited***, ***unedited*** and ***unintended edit*** proportions. A formal description of each component of the model is described in their respective sections below.

1.2 Sequence encoder network

1.2.1 Embedding block

Formally, given a design variant sequence (mutated or original sequence) $\underline{S} = [x_1, x_2, \cdots, x_T]$, a nucleotide at position t is represented by 1-of-K encoding where K is the size of the set of all nucleotide letters in the data such that $x_t \in [0,1]^K$ and T is the length of the sequence. An embedding matrix W_e is used to map the input x_t to a fixed-length vector representation (Eq. 1)

$$e_t = W_e x_t \tag{1}$$

where $W_e \in \mathbb{R}^{d_e \times K}$, $e_t \in \mathbb{R}^{d_e}$, and d_e is the dimension of vector e_t.
Similarly, each position p_t in the sequence $\underline{S}$ has multiple set of annotations indicating if it belongs to protospacer, PBS, and RTT (denoted by p_t^{proto}, p_t^{PBS}, and p_t^{RTT} respectively). These annotations are represented by binary indicators where p_t^{proto}, p_t^{PBS}, and $p_t^{RTT} \in [0,1]$. An embedding matrix corresponding to each annotation $W_{p^{proto}}$, $W_{p^{PBS}}$, $W_{p^{RTT}}$ is used to map the binary indicators to a fixed-length vector representation (Eq. 2)

$$\begin{aligned} a_t^{proto} &= W_{p^{proto}} p_t^{proto} \\ a_t^{PBS} &= W_{p^{PBS}} p_t^{PBS} \\ a_t^{RTT} &= W_{p^{RTT}} p_t^{RTT} \end{aligned} \tag{2}$$

where $W_{p^{proto}}$, $W_{p^{PBS}}$, $W_{p^{RTT}} \in \mathbb{R}^{d_a \times 2}$, a_t^{proto}, a_t^{PBS}, $a_t^{RTT} \in \mathbb{R}^{d_a}$ and d_a is the annotation embedding dimension.

Computed embeddings at each position were concatenated $\bigoplus$ (Eq. 3) to get a unified representation for every token/element in the sequence $\underline{S}$ (i.e. compute a new sequence $\underline{U} = [u_1, u_2, \cdots, u_T]$ where $u_t \in \mathbb{R}^{d_u}$, $\forall t \in [1, \cdots, T]$ and $d_u = d_e + 3 \times d_a$ for the original sequence and $d_u = d_e + 2 \times d_a$ for the mutated sequence).

$$u_t = [e_t \bigoplus a_t^{proto} \bigoplus a_t^{PBS} \bigoplus a_t^{RTT}] \tag{3}$$

1.2.2 Encoder block: bidirectional GRU

The computed embeddings of sequence $\underline{U} = [u_1, u_2, \cdots, u_T]$ from the embedding block is further processed using a recurrent neural network (RNN) encoder block. A vanilla RNN will compute a hidden vector at each position (i.e. state vector h_t at position t), representing a history or context summary of the sequence using the input and hidden states vector form the previous steps. Equation 4 shows the computation of the hidden vector h_t using the input u_t and the previous hidden vector h_{t-1} where ϕ is a non-linear transformation such as $ReLU(z) = max(0, z)$ or $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.

$$h_t = \phi(W_{hu} u_t + W_{hh} h_{t-1} + b_{hu}) \tag{4}$$

$W_{hh} \in \mathbb{R}^{d_h \times d_h}$, $W_{hu} \in \mathbb{R}^{d_h \times d_u}$, $b_{hu} \in \mathbb{R}^{d_h}$, represent the RNN's weights to be optimized and d_h, d_u are the dimensions of h_t and u_t vectors respectively. Note that the weights are shared across the network. The use of RNN allows the model to learn long-range dependencies where the network is unfolded as many times as the length of the sequence (i.e. length of original and mutated sequences) it is modeling. Although RNNs are capable of handling and representing variable-length sequences, in practice, the learning process faces challenges due to the vanishing/exploding gradient problem [4–6]. In this work, we used gated recurrent unit (GRU) [2,3] to overcome the latter challenges by updating the computation mechanism of the hidden state vector h_t through the specified equations below.

$$\begin{aligned}
z_t &= \sigma(W_{hu}^z u_t + W_{hh}^z h_{t-1} + b_{hu}^z) && \text{(update gate)} \\
r_t &= \sigma(W_{hu}^r u_t + W_{hh}^r h_{t-1} + b_{hu}^r) && \text{(reset gate)} \\
\tilde{h}_t &= \phi(W_{hu}^{\tilde{h}} u_t + r_t \odot W_{hh}^{\tilde{h}} h_{t-1} + b_{hu}^{\tilde{h}}) && \text{(new state/memory cell)} \\
h_t &= (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1} && \text{(hidden state vector)}
\end{aligned}$$

The GRU model computes a reset gate r_t that is used to modulate the effect of the previous hidden state vector h_{t-1} when computing the new memory vector $\tilde{h}_t$. The update gate z_t determines the importance/contribution of the newly generated memory vector $\tilde{h}_t$ compared to the previous hidden state vector h_{t-1} when computing the current hidden vector h_t. The weights W_{hu}^z, W_{hu}^r, $W_{hu}^{\tilde{h}}$ each $\in \mathbb{R}^{d_h \times d_u}$ and W_{hh}^z, W_{hh}^r, $W_{hh}^{\tilde{h}}$ each $\in \mathbb{R}^{d_h \times d_h}$. The biases b_{hu}^z, b_{hu}^r, $b_{hu}^{\tilde{h}}$ each $\in \mathbb{R}^{d_h}$ where d_h and d_u are the dimensions of h_t and u_t vectors respectively. The operator σ represents the *sigmoid* function, ϕ the *tanh* or *ReLU* function, and $\odot$ the element-wise product (i.e. Hadamard product).

In our work we used a bidirectional GRU that computes two hidden state vectors $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ for each element u_t in sequence $\underline{U}$ corresponding to left-to-right and right-to-left GRU encoding of the sequence at position t. Hence, the newly computed sequence $\underline{G} = [\overleftrightarrow{g_1}, \overleftrightarrow{g_2}, \cdots, \overleftrightarrow{g_T}]$ is based on the *concatenation* of the computed hidden vector representations at each position t (see Eq. 5)

$$\overleftrightarrow{g_t} = [\overrightarrow{h_t} \bigoplus \overleftarrow{h_t}] \tag{5}$$

where $\overleftrightarrow{g_t} \in \mathbb{R}^{d_g}$ and $d_g = 2 \times d_h$.

1.2.3 Attention layer block

The attention layer served as a *pooling* or *compression* layer where the sequence of learned representations $\underline{G}$ are mapped to one fixed-vector representation. We adapted the idea of a *global* attention model [7,8] in which a global context/query vector q_{global} (i.e. trainable parameters) was used along with the output $\underline{G} = [\overleftrightarrow{g_1}, \overleftrightarrow{g_2}, \cdots, \overleftrightarrow{g_T}]$ from the sequence encoder block to generate a pooled representation vector z_{global} (Eq. 6). The objective is to compute *attention* weights α_t for every $\overleftrightarrow{g_t}$ vector where α_t is the normalized weight computed using Eq. 7.

$$z_{global} = \sum_{t=1}^{T} \alpha_t \overleftrightarrow{g_t} \tag{6}$$

$$\alpha_t = \frac{\exp\left(score(q_{global}, \overleftrightarrow{g_t})\right)}{\sum_{k=1}^{T} \exp\left(score(q_{global}, \overleftrightarrow{g_k})\right)} \tag{7}$$

Similarly, we used another attention pooling layer comprising of a trainable context vector q_{RTT} (i.e. trainable parameters) focused on the RTT region of the sequence where mutations are encoded. As a result, the attention weights were computed only based on the elements belonging to the RTT region and later were used to generate a weighted representation z_{RTT} of these elements

$$z_{RTT} = \sum_{t=1}^{T} (\alpha_t \overleftrightarrow{g_t}) \cdot \mathbb{1}[p_t^{RTT} = 1] \tag{8}$$

where α_t is the normalized weight computed using Eq. 9, and $\mathbb{1}[p_t^{RTT} = 1]$ is an indicator variable equal to 1 when the element at position t is part of the RTT region and 0 otherwise.

$$\alpha_t = \frac{\exp\left(score(q_{RTT}, \overleftrightarrow{g_t}, \mathbb{1}[p_t^{RTT} = 1])\right)}{\sum_{k=1}^{T} \exp\left(score(q_{RTT}, \overleftrightarrow{g_k}, \mathbb{1}[p_k^{RTT} = 1])\right)\cdot} \tag{9}$$

For the attention *scoring* function, we defined *score* using the scaled dot-product as in [9] (see Equations 10 and 11 corresponding to the global vector q_{global} and RTT vector q_{RTT} respectively). In both equations, the score is computed by performing a *dot-product* between the query vector q and $\overleftrightarrow{g_t}$ scaled by d_g which is the dimension of both vectors.

$$score(q_{global}, \overleftrightarrow{g_t}) = \frac{q_{global}^{\top} \cdot \overleftrightarrow{g_t}}{\sqrt{d_g}} \tag{10}$$

$$score(q_{RTT}, \overleftrightarrow{g_t}, \mathbb{1}[p_t^{RTT} = 1]) = \frac{q_{RTT}^{\top} \cdot \overleftrightarrow{g_t}}{\sqrt{d_g}} \cdot \mathbb{1}[p_t^{RTT} = 1] \tag{11}$$

After the attention layer operations, the sequence encoder block generates a pair of vectors z_{global} and z_{RTT} for each of the original and mutated sequence separately (i.e. totaling four vectors) that are stacked to create z_{seq} (Eq. 12)

$$z_{seq} = [z_{global}^{original} \bigoplus z_{RTT}^{original} \bigoplus z_{global}^{mutated} \bigoplus z_{RTT}^{mutated}] \tag{12}$$

1.3 Sequence-level features encoder network

In addition to using the original and mutated sequences, we further derived *sequence-level features* such as the melting temperature or minimum free energy for the different parts of the sequences. We derived 24 features that were passed as an input vector to a *sequence-level encoder* comprising a stack of multilayer feed-forward networks, residual connections and layer-normalization operations to map the input features to a dense vector representation. Formally, an input vector $f_{seq} \in \mathbb{R}^{d_f}$ representing *sequence-level features* is mapped using an affine transformation matrix (Eq. 13)

$$f'_{seq} = W_f f_{seq} + b_f \tag{13}$$

where $W_f \in \mathbb{R}^{d' \times d_f}$, $b_f \in \mathbb{R}^{d'}$, d_f, and d' representing the dimensions of f_{seq} and f'_{seq} vectors respectively. The embedded vector f'_{seq} is later passed to a feed-forward network consisting of two affine transformation matrices and non-linear activation function to further compute/embed a new vector representation. The first transformation (Eq. 14) uses $W_{MLP1} \in \mathbb{R}^{\xi d' \times d'}$ and $b_{MLP1} \in \mathbb{R}^{\xi d'}$ to transform input f'_{seq} to new vector $\in \mathbb{R}^{\xi d'}$ where $\xi \in \mathbb{N}$ is multiplicative factor. A non-linear function such as $ReLU(z) = max(0, z)$ is applied followed by another affine transformation using $W_{MLP2} \in \mathbb{R}^{d' \times \xi d'}$ and $b_{MLP2} \in \mathbb{R}^{d'}$ to obtain vector $\tilde{f}_{seq} \in \mathbb{R}^{d'}$. A layer normalization (Eq. 15) is applied to obtain $\tilde{f}'_{seq} \in \mathbb{R}^{d'}$.

$$\tilde{f}_{seq} = W_{MLP2} ReLU(W_{MLP1} f'_{seq} + b_{MLP1}) + b_{MLP2} \tag{14}$$

$$\tilde{f}'_{seq} = LayerNorm(\tilde{f}_{seq}) \tag{15}$$

Layer normalization [10] was used with the goal to ameliorate the "covariate-shift" problem by re-standardizing

the computed vector representations (i.e. using the mean and variance across the features/embedding dimension d'). Given a computed vector $\tilde{f}_{seq}$, $LayerNorm$ function will standardize the input vector using the mean μ and variance σ^2 along the features dimension d' and apply a scaling γ and shifting step β (Eq. 18). γ and β are learnable parameters and ϵ is small number added for numerical stability.

$$\mu = \frac{1}{d'} \sum_{j=1}^{d'} \tilde{f}_{seq(j)} \tag{16}$$

$$\sigma^2 = \frac{1}{d'} \sum_{j=1}^{d'} (\tilde{f}_{seq(j)} - \mu)^2 \tag{17}$$

$$LayerNorm(\tilde{f}_{seq}) = \gamma \times \frac{\tilde{f}_{seq} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{18}$$

Furthermore, we used residual connections / skip-connections [11] in order to improve the gradient flow in layers during training. This is done by summing both the newly computed output of the current layer with the output from the previous layer. In our setting, a residual connection is applied by summing f'_{seq} vector (first embedded vector) with the $\tilde{f}_{seq}$ vector and then passed to a layer normalization layer.

1.4 Decoder block

The last layer in the model takes as input the computed representation vectors from the two **sequence encoder blocks** (original and mutated sequence) z_{seq} (Eq. 12), and the **sequence-level features encoder block** $\tilde{f}'_{seq}$. These vectors are concatenated to become one input vector z_{final}

$$z_{final} = [z^{original}_{global} \bigoplus z^{original}_{RTT} \bigoplus z^{mutated}_{global} \bigoplus z^{mutated}_{RTT} \bigoplus \tilde{f}'_{seq}] \tag{19}$$

z_{final} is passed through a stack of multilayer feed-forward networks, residual connections and layer-normalization operations to compute a probability distribution on the outcomes (i.e. **edited**, **unedited**, **unintended edits**). An affine transformation matrix is first applied (Eq. 20)

$$z'_{final} = W_z z_{final} + b_z \tag{20}$$

where $W_z \in \mathbb{R}^{d' \times d_z}$, $b_z \in \mathbb{R}^{d'}$, d_z, and d' representing the dimensions of z_{final} and z'_{final} vectors respectively. The embedded vector z'_{final} is later passed to a feed-forward network consisting of two affine transformation matrices and non-linear activation function to further compute/embed a new vector representation. The first transformation (Eq. 21) uses $W^z_{MLP1} \in \mathbb{R}^{\xi d' \times d'}$ and $b^z_{MLP1} \in \mathbb{R}^{\xi d'}$ to transform input z'_{final} to new vector $\in \mathbb{R}^{\xi d'}$ where $\xi \in \mathbb{N}$ is multiplicative factor. A non-linear function such as $ReLU(z) = max(0, z)$ is applied followed by another affine transformation using $W^z_{MLP2} \in \mathbb{R}^{d' \times \xi d'}$ and $b^z_{MLP2} \in \mathbb{R}^{d'}$ to obtain vector $\tilde{z}_{final} \in \mathbb{R}^{d'}$. A layer normalization (Eq. 22) is applied to obtain $\tilde{z}'_{final} \in \mathbb{R}^{d'}$.

$$\tilde{z}_{final} = W^z_{MLP2} ReLU(W^z_{MLP1} z'_{final} + b^z_{MLP1}) + b^z_{MLP2} \tag{21}$$

$$\tilde{z}'_{final} = LayerNorm(\tilde{z}_{final}) \tag{22}$$

A last affine transformation is applied to $\tilde{z}'_{final}$ followed by $softmax$ operation to compute a probability distribution on the outcomes (i.e. three outcomes in our current setting). That is, the probability distribution $\hat{y}$ is computed using Eq. 23

$$\hat{y} = \sigma(W_o \tilde{z}'_{final} + b_o) \tag{23}$$

where $\hat{y} \in \mathbb{R}^{|Y|}$, $W_o \in \mathbb{R}^{|Y| \times d'}$, $b_o \in \mathbb{R}^{|Y|}$, Y is the set of admissible outcomes, $|Y|$ is the number of outcomes (i.e. proportions of three outcomes in our case), d' is the dimension of $\tilde{z}'_{final}$ and σ is the $softmax$ function.

1.5 Objective Function

We used Kullback–Leibler divergence (D_{KL}) as a loss function for an i-th design variant sample (i.e. pair of original and mutated sequences). D_{KL} represents the relative entropy of the model's estimated distribution over all target outcomes $\hat{y}$ with respect to the true distribution y (i.e. observed proportions of edited, unedited and unintended edits) (Eq. 24).

$$D^i_{KL}(y^i||\hat{y}^i) = \sum_{j=1}^{|Y|} y^i_j log(\frac{y^i_j}{\hat{y}^i_j}) \tag{24}$$

Lastly, the objective function for the whole training set is defined by the average loss across all the design variant sequences plus a weight regularization term λ (i.e. l_2-norm regularization) applied to the model parameters represented by $\boldsymbol{\theta}$

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} D^i_{KL}(y^i||\hat{y}^i) + \frac{\lambda}{2}||\boldsymbol{\theta}||^2_2 \tag{25}$$

In practice, the training occurs using mini-batches where computing the loss function and updating the parameters/weight occur after processing each mini-batch of the training set.

Supplementary Methods 1b: Experiments for machine learning

2.1 Training & Evaluation Workflow

Our dataset included of 92,925 design variants (see Fig. 1 and Suppl. Fig 1 in the main manuscript describing the library design) that were used to train and evaluate machine learning models. We followed a grouped 5-fold cross-validation where the grouping was based on disease locus (pegRNAs for the same locus were not split between training-, test- and validation sequences). Each fold had 80% of the sequences for training and 20% for testing. A validation set for each fold was created by taking a 10% grouped random split (again grouping was based on disease locus) from the fold's training sequences. For each fold, a model was trained on the training sequences and then evaluated on the corresponding test sequences of that fold.

2.1.1 PRIDICT Model

PRIDICT model performance was evaluated using Pearson and Spearman correlation. During models' training, the epoch in which the model achieved the best harmonic mean between both scores on the validation set was recorded, and model state as it was trained up to that epoch was saved. This best model, as determined by the validation set, was then tested on the test split. The evaluation of the trained models was based on their average performance on the test sets across the five folds.

2.2 Hyperparameters Optimization

We used a uniform random search strategy [8, 12] that randomly chose a set of hyperparameters configurations (i.e. embedding dimension, number of hidden layers, dropout probability, etc.) from the set of all possible configurations and trained corresponding models on one chosen random fold (i.e. one fold out of the 5-folds and fixed for all models). Then the best configuration for each model (i.e. the one achieving best performance on the validation set) was used for the final training and testing of each model on all 5 folds.
The range of possible hyperparameters configuration (i.e. choice of values for hyperparameters) for PRIDICT models is reported in Table 1.

List 1 PRIDICT hyperparameters options

- Embedding Block operations
  - Nucleotide embedding layer
    - embedding dimension d' ..... $\{16, 32, 64, 128\}$
  - Annotation embedding layer
    - embedding dimension d_a ..... $\{4, 8, 16\}$
  - concatenate option ..... $\{stacking, adding\}$
- Encoder Block operations
  - GRU
    - number of hidden layers ..... $\{1, 2\}$
    - hidden dimension ..... $\{16, 32, 64, 128\}$
    - embedding dimension ..... $\{16, 32, 64, 128\}$
    - dropout ..... $\{0.15, 0.25, 0.35\}$
  - Sequence-level features encoder
    - Feed-Forward (MLP) block
      - embedding dimension ..... $\{16, 32, 64, 128\}$
      - MLP embedding factor (multiplier) ξ ..... $\{1, 2, 3\}$
      - Non-linear function ..... $\{ELU, ReLU\}$
      - dropout ..... $\{0.15, 0.25, 0.35\}$
    - number of repeats for MLP Block ..... $\{1, 2\}$
- Decoder Block operations
  - Feed-Forward (MLP) block
    - embedding dimension ..... $\{16, 32, 64, 128\}$
    - MLP embedding factor (multiplier) ξ ..... $\{1, 2, 3\}$
    - Non-linear function ..... $\{ELU, ReLU\}$
    - dropout ..... $\{0.15, 0.25, 0.35\}$
  - number of repeats for MLP Block ..... $\{1, 2\}$
- l_2-norm regularization λ ..... $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$
- Batch size during training ..... $\{1000, 1500, 2000\}$
- Optimization algorithm ..... $\{Adam\}$

Supplementary Methods 1c: Feature importance

3.1 Integrated gradients

We adapted integrated gradients (IG) approach [13] to evaluate the contribution of the different features used in PRIDICT model including the implicit ones such as the nucleotide and position annotations in the original and mutated sequences, and the explicit ones based on the derived sequence-level features. IG belongs to a gradient-based (perturbation) class of methods for analyzing feature importance. For every sample input under consideration, a baseline (such as zero input baseline) is chosen as starting point and then a set of interpolated samples are generated on a linear path based on the distance between the initial baseline and the current input sample representation in the embedding space. These interpolated samples represent a set of perturbations (along a linear path) to reach from the baseline to the current input sample representation in the embedding space. These perturbed samples are fed to the model to predict their target outcome (i.e., edit or unintended edit proportion) and then the gradient of the output with respect to these inputs is computed. By integrating the computed gradients for the set of perturbations and then rescaling it by the distance between the current input sample and the perturbed baseline, we can evaluate the importance of each feature in the input sample on the outcome prediction (see Eq. 26 and 27). By repeating this process to each sample in the dataset (i.e., test set), we can evaluate the overall importance of the features when using PRIDICT for efficiency prediction.

$$IG_i(x) = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha \tag{26}$$

$$IG_i(x) = (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m} \tag{27}$$

4 References

[1] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. Devito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[2] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734. [Online]. Available: http://aclweb.org/anthology/D14-1179

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," 2014.

[4] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, nov 1997. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735

[5] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, mar 1994. [Online]. Available: http://ieeexplore.ieee.org/document/279181/

[6] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, ser. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 385. [Online]. Available: http://link.springer.com/10.1007/978-3-642-24797-2

[7] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," aug 2015. [Online]. Available: http://arxiv.org/abs/1508.04025

[8] K. F. Marquart, A. Allam, S. Janjuha, A. Sintsova, L. Villiger, N. Frey, M. Krauthammer, and G. Schwank, "Predicting base editing outcomes with an attention-based deep learning algorithm trained on high-throughput target library screens," *Nature Communications 2021 12:1*, vol. 12, no. 1, pp. 1–9, aug 2021. [Online]. Available: https://www.nature.com/articles/s41467-021-25375-z

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," jun 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[10] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," jul 2016. [Online]. Available: http://arxiv.org/abs/1607.06450

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December. IEEE Computer Society, dec 2016, pp. 770–778.

[12] J. Bergstra and Y. Bengio, "Random Search for HyperParameter Optimization," *Journal of Machine Learning Research*, 2012.

[13] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," 2017.