Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTERTHESIS

Fehlerfortpflanzung in RNA-seq Analyse

vorgelegt von

Marcos Ivan Chow Castro

MIN-Fakultät

Fachbereich Informatik

Zentrum für Bioinformatik

Studiengang: M.Sc. Bioinformatik

Matrikelnummer: 7386285

Ggf. Abgabedatum: 08.10.2025

Erstgutachter: Prof. Dr. Andrew Torda

Zweitgutachter: Dr. Jannek Squar

Zusammenfassung

RNA-Seq-Pipelines bestehen aus mehreren Schritten, in denen sich Unsicherheiten fortpflanzen. Diese Fallstudie isoliert gezielt kleine, zufällige *Sequenzierfehler* und misst, wie stark sie sich bis zu DE- und GSEA-Ergebnissen durchschlagen. Methodisch unterscheiden wir drei Grundfälle der Fehlerfortpflanzung: (i) Fortpflanzung durch Funktionen via Linearisierung (Delta-Methode), (ii) quadratische Kombination unabhängiger Beiträge (absolute bzw. relative Unsicherheiten) und (iii) Kombination abhängiger Beiträge über Kovarianzen [4]. Für stark nichtlineare oder implizite Modelle empfiehlt sich eine *Monte-Carlo*-Propagation, eine simulationsbasierte, stochastische Methode, bei der man zufällige Stichproben aus den Unsicherheitsverteilungen der Eingangsgrößen zieht, die vollständige Analyse für jede Stichprobe ausführt und so die empirische Verteilung der Ausgabegrößen erhält [4]. Auf Basis von GSE52778 (4 Spender, 4 Bedingungen) lief ein automatisierter Workflow (Trim Galore!, STAR, featureCounts, DESeq2 mit dem Modell `~ donor + condition`). In die FASTQ-Dateien wurden ausschließlich *Basensubstitutionen* mit per-Base-Raten von *0.01 %, 0.05 %, 0.10 %* injiziert (jeweils $N = 1000$), die fehlerfreie Analyse dient als Referenz. *Global* bleiben PCA-Struktur und LFCs sehr stabil (Pearson/Spearman ≥ 0.95) ohne systematischen Bias (Bland-Altman). *Lokal* zeigt sich eine ausgeprägte Schwellenfragilität, dass die Anzahl der p-Wert mit der Fehlerrate steigt und dass der Jaccard-Index zwischen DEG-Listen dosisabhängig abnimmt. Diese Schwellenfragilität wirkt sich bis auf die funktionelle Ebene aus. Während übergeordnete GSEA-Themen stabil bleiben, verschieben sich einzelne GO-Terme in ihrer Rangfolge. Die Ergebnisse unterstreichen daher die Notwendigkeit, DEG-Listen durch Stabilitätsmaße wie den Jaccard-Index oder Bland-Altman-Analysen zu kontextualisieren. Für künftige Simulationen empfiehlt es sich, das Modell um weitere Faktoren wie Quantifizierungsunschärfen und Sequenzabdeckung zu erweitern.

Abkürzungsverzeichnis

BAM Binary Alignment Map

DEG Differentiell exprimiertes Gen

DGE Differentielle Genexpression

FASTQ Textbasiertes Format zur Speicherung von Sequenzdaten und Qualitätswerten

GO Gene Ontology

GSEA Gene Set Enrichment Analysis

LFC Log2-Fold-Change

PCA Hauptkomponentenanalyse (Principal Component Analysis)

SAM Sequence Alignment Map

SE Standardfehler (Standard Error)

SRA Sequence Read Archive

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Hochdurchsatz-Sequenzierung von Ribonukleinsäure (RNA-Seq) hat sich als Goldstandard für die quantitative Analyse von Transkriptomen etabliert [43]. Ihre Fähigkeit, ein umfassendes und quantitatives Bild der Genexpression zu liefern, hat die biologische und medizinische Forschung revolutioniert von der Grundlagenforschung bis hin zur klinischen Diagnostik. Komplexe bioinformatische Pipelines sind jedoch erforderlich, um aus den rohen Sequenzierdaten biologisch interpretierbare Ergebnisse zu extrahieren. Während einige Schritte dieser Pipelines, wie beispielsweise die Konvertierung von Dateiformaten, deterministisch sind, sind andere, wie das Alignment der Reads oder die Quantifizierung der Genexpression, anfällig für Fehler und Mehrdeutigkeiten. Die Validität der finalen Ergebnisse hängt entscheidend von der Genauigkeit jedes einzelnen vorgeschalteten Schrittes ab, was die Untersuchung potenzieller Fehlerquellen zu einem zentralen Anliegen der Bioinformatik macht. Aktuelle Übersichtsarbeiten fassen den methodischen Reifegrad von RNA-seq und typische Fehlerquellen zusammen [40]. Nicht alle Schritte sind gleich fehleranfällig, während Formatkonvertierungen deterministisch sind, akkumulieren bei *Alignment* (Mehrdeutigkeiten, Referenzbias), *Quantifizierung* (Feature-Überlappungen, Multi-Mapping) und *Statistik* (Dispersion, FDR) Unsicherheiten. Diese Fallstudie isoliert *zufällige Base-Call-Fehler* als frühe Quelle und prüft, inwieweit deren Effekte bis zur Signifikanzschwelle und GSEA propagieren. Großangelegte Reproduzierbarkeitsstudien belegen, dass DEG-Listen und Rangfolgen sensibel auf technische/analytische Entscheidungen reagieren [36].

1.2 Grundlagen

Eine typische RNA-Seq-Analyse beginnt mit der Verarbeitung von Rohdaten im FASTQ-Format [8]. Diese werden mittels eines Spliced Transcripts Alignments wie `STAR` [10] an ein Referenzgenom angebunden. Anschließend werden die alignierten Reads quantifiziert, um eine Zählmatrix zu erstellen, die für jedes Gen die Anzahl der zugeordneten Reads pro Probe enthält. Werkzeuge wie `featureCounts` [21] werden hierfür häufig verwendet. Die statistische Analyse zur Identifizierung differentiell exprimierter Gene (DGE) wird oft mit spezialisierten Paketen wie `DESeq2` [22] durchgeführt. `DESeq2` modelliert die Zähldaten mithilfe einer negativen Binomialverteilung, um die biologische und technische Varianz adäquat zu berücksichtigen. Das Ergebnis ist eine Liste von Genen, deren Expression sich zwischen den experimentellen Bedingungen signifikant ändert, typischerweise bewertet durch den $\log_2$ Fold Change, p-Wert und einen adjustierten p-Wert. Nachgelagerte Analysen wie die Gene Set Enrichment Analysis (GSEA) [41] nutzen diese Genlisten, um angereicherte biologische Signalwege zu identifizieren und so eine funktionelle Interpretation der Ergebnisse zu ermöglichen. (*Gene Ontology, GO* [1])

1.3 Problemstellung

Die einzelnen Schritte der beschriebenen Pipeline sind in unterschiedlichem Maße anfällig für Fehler. Fehler können bereits in den ursprünglichen Rohdaten vorhanden sein, beispielsweise durch den

Sequenzierprozess selbst (`Sequenzierfehler`), oder während der bioinformatischen Verarbeitung entstehen, etwa bei der Zuweisung von Reads zu Genen (`Quantifizierungsfehler`). Solche Fehler, die sich als fehlerhafte Basenaufrufe in den FASTQ-Dateien oder als fehlerhafte Zuordnungen von Reads manifestieren, haben das Potenzial, sich durch die nachfolgenden Analyseschritte fortzupflanzen ein Phänomen, das als Fehlerfortpflanzung (engl. *error propagation*) bekannt ist. Der Begriff wird klassisch über analytische Fortpflanzungsformeln (Delta-Methode) gefasst [19, 28] und in praxisnahen Leitfäden zur Unsicherheitsbewertung systematisiert [42]. Falsche Basen können zu Fehlern im Alignment führen (mismatches), was die Quantifizierung beeinflusst und letztlich die statistischen Ergebnisse verfälschen kann.
Während moderne Illumina-Sequenzierer typischerweise eine sehr niedrige durchschnittliche Fehlerrate von unter 0,01 % aufweisen [31], ist diese Rate nicht uniform. Sie kann durch technische Probleme lokal erhöht sein. Es ist daher weitgehend unklar, in welchem Ausmaß die statistischen Modelle der DGE-Analyse robust gegenüber solchem Rauschen sind. Eine wichtige Problemstellung dieser Arbeit ist daher die Quantifizierung der Auswirkungen von initialen Sequenzierfehlern auf die Stabilität und Verlässlichkeit der finalen biologischen Schlussfolgerungen einer RNA-Seq-Analyse.

1.4 State of the Art

Die Robustheit von RNA-Seq-Auswertepipelines ist intensiv untersucht. Großprojekte wie SEQC/MAQC-III zeigten, dass Plattform- und Methodenauswahl die Messgenauigkeit und Reproduzierbarkeit wesentlich beeinflusst [36]. Systematische Vergleiche von Alignern und Quantifizierungstools berichten deutliche Leistungsunterschiede und methodenspezifische Sensitivitäten [11, 34]. Zudem ist die Entdeckungsrate und FDR-Kontrolle stark von der Zahl biologischer Replikate abhängig [35]. Jüngere Arbeiten belegen zudem, dass bereits die Wahl der Mapping/Alignment-Strategie die Quantifizierung und nachgelagerte DE-Analysen merklich verändert [39]. Während existierende Studien die Fehlerraten der Sequenzierung quantifiziert haben, fehlt jedoch häufig eine systematische Isolierung einer variablen Anfangs-Fehlerrate und deren Propagation bis hin zur funktionellen Interpretation eine Lücke, die diese Arbeit mit kontrollierten Simulationen adressiert [31, 36].

1.5 Ziel der Arbeit

Die Verlässlichkeit von RNA-Sequencing-Analysen (RNA-Seq) hängt entscheidend von der Qualität der zugrundeliegenden Rohdaten ab. Sequenzierartefakte und Fehler können sich durch die mehrstufige bioinformatische Analyse-Pipeline fortpflanzen und die finalen Ergebnisse potenziell verfälschen. Diese Arbeit untersucht diese Fehlerfortpflanzung systematisch im Rahmen einer Fallstudie. Ziel ist es, ihre quantitativen und qualitativen Auswirkungen auf die biologische Interpretation zu bewerten.
Das zentrale Ziel dieser Arbeit ist es, die Hypothese zu überprüfen, dass *kleine, zufällige Sequenzierfehler sich durch die Pipeline fortpflanzen und die Robustheit sowie Reproduzierbarkeit der finalen Ergebnisse beeinflussen*. Für die Simulationen verwendeten wir die öffentlichen Rohdaten des NCBI Gene Expression Omnibus (GEO), Serien-Accession GSE52778, sowie die zugehörigen Rohläufe im NCBI SRA/BioProject (PRJNA229998). [12, 32] Hierbei werden in die FASTQ-Rohdaten gezielt `ausschließlich Sequenzierfehler` mit definierten Raten (0.01 %, 0.05 % und 0.10 %) eingeführt. Andere potenzielle Fehlerquellen wie Quantifizierungsfehler werden in dieser Studie nicht simuliert. Die Untersuchung konzentriert sich auf die Beantwortung der folgenden zentralen Forschungsfrage:

> *Welchen Einfluss hat die Fortpflanzung von Sequenzierfehlern auf die Stabilität und Reproduzierbarkeit der Listen signifikant differentiell exprimierter Gene (DEGs)?*

Zur Beantwortung dieser Frage werden die Ergebnisse der fehlerbehafteten Analysen systematisch mit einer fehlerfreien Referenzanalyse auf vier komplementären Ebenen verglichen:

1. Globale Datenstruktur: Veränderung der Proben-Clusterung in der Hauptkomponentenanalyse (PCA).
2. Quantitative Effektstärken: Robustheit der berechneten $\log_2$ Fold Changes.
3. Statistische Stabilität: Konsistenz der p-Werte und Übereinstimmung der DEG-Listen.
4. Biologische Interpretation: Übereinstimmung der Ergebnisse der Gene Set Enrichment Analysis (GSEA).

Finales Ziel ist es, eine fundierte Aussage über die Robustheit des DGE-Workflows zu treffen, kritische, fehleranfällige Punkte in der Analyse zu identifizieren und somit die fundamentale Bedeutung der initialen Datenqualität für die Validität von Transkriptomstudien zu untermauern.

2 Methodik

2.1 Begriffsrahmen und Typen der Fehlerfortpflanzung

Zur Quantifizierung technischer Unsicherheiten existieren *experimentelle* Kontrollstrategien und *in-silico* Ansätze. Zu den experimentellen zählen insbesondere *Spike-in*-Kontrollen (synthetische RNA-Standards wie ERCC, SIRV, Sequins) zur Prüfung von Linearität, Dynamikbereich/LOD und technischer Varianz sowie zum Benchmarking kompletter Workflows [17, 13, 37]. In dieser Arbeit werden *keine* Spike-ins eingesetzt. Sie sind hier nur der Vollständigkeit halber erwähnt. Praktische Gründe: (i) *direkte Materialkosten* für Spike-in-Mixe/Kits, (ii) *Sequenzierkosten* durch belegte Reads (oder zusätzliche Lanes), (iii) *Experimentdesign-Overhead* (geeignete Konzentrationsspannen, Replikate), (iv) *Auswertungsaufwand* (Kalibrierung/Integration). Außerdem verhalten sich exogene Standards nicht immer wie endogene Transkripte (GC-/Fragmentierungs-/Poly-A-Effekte), was die Interpretation erschwert [17, 13]. Weitere verbreitete Alternativen sind *technische Replikate* (direkt, aber teuer in Probenzahl/Sequenzkapazität), *UMIs* (präzisere Zählungen, jedoch spezielle Library-Preps und zusätzliche Bioinformatik), *qPCR-Validierungen* ausgewählter Targets (gezielt, aber punktuell) sowie *Verdünnungsreihen* mit synthetischem Input. Diese Arbeit fokussiert sich bewusst auf einen *in-silico* Zugang via Monte-Carlo.

Wir unterscheiden *Fehler* von *Unsicherheit*, systematische Abweichungen (Bias) vs. zufällige Streuung. Erstere verzerren Erwartungswerte, letztere erhöhen Varianzen. Zusätzlich trennen wir *epistemische* Unsicherheit (wissensbedingt, prinzipiell reduzierbar) und *aleatorische* Unsicherheit (inhärente Variabilität) [9]. In RNA-Seq sind Sequenzierfehler primär aleatorisch. Unvollständige Referenzen/Annotationen erzeugen epistemische Komponenten.

Fehlerquellen entlang der Pipeline: (a) *Sequenzierung* (Base-Call-Fehler, Positions-/Kontexteffekte), (b) *Alignment* (Mehrdeutigkeiten, Referenzbias), (c) *Quantifizierung* (Feature-Überlappungen, Multi-Mapping) und (d) *Statistik* (Dispersion, Mehrfachtest). Diese Studie isoliert (a) und verfolgt deren Fortpflanzung bis zu DGE/GSEA. Die Pipeline besteht aus Trim Galore! $\rightarrow$ STAR $\rightarrow$ featureCounts $\rightarrow$ DESeq2 [10, 21, 22].

Für ein Ergebnis $y = f(\mathbf{x})$ mit Eingaben $\mathbf{x}$ (Mittelwert $\boldsymbol{\mu}$, Kovarianz Σ) liefert die lineare Approximation (Delta-Methode) die Varianzabschätzung

$$\mathrm{Var}[f(\mathbf{x})] \approx \nabla f(\boldsymbol{\mu})^{\top} \, \Sigma \, \nabla f(\boldsymbol{\mu})$$

[1]

Bei Summen addieren sich Varianzen, bei Produkten addieren sich *relative* Varianzen. Abhängigkeiten gehen über Kovarianzen ein. Für stark nichtlineare oder implizite f empfiehlt sich eine *Monte-Carlo-Fortpflanzung*, die Eingangsunsicherheiten simulativ durch das Modell trägt und die Ausgabeverteilung empirisch schätzt [19, 28, 16, 42, 4].

Die RNA-Seq-Pipeline ist deterministisch, aber hochgradig nichtlinear und mehrstufig, eine analytische Ableitung von ∇f ist weder praktikabel noch stabil. Daher verwenden wir Monte-Carlo-Fortpflanzung: exogene Störung der FASTQ-Eingaben (substitutionsbasierte Fehler mit Rate ε), vollständiger Pipeline-Durchlauf, Auswertung vordefinierter Metriken relativ zur Referenz. So erhalten wir direkt die Verteilun-

[1] Im sogenannten Numerator-Layout wird der Gradient als Zeilenvektor notiert; dann lautet die äquivalente Quadratikform $\nabla f(\boldsymbol{\mu}) \, \Sigma \, \nabla f(\boldsymbol{\mu})^{\top}$. Vgl. [25, 30, 15].

gen relevanter Größen (u. a. LFC-Kohärenz, Bland-Altman-Bias/LoA, Jaccard-Index, GSEA-Kohärenz) über N Wiederholungen und Fehlerraten.
Wir modellieren aleatorische Sequenzierfehler als unabhängige Bernoulli-verteilte Ereignisse mit per-Base-Rate ε und gleichverteilter Substitutionsbasis, d. h. bei einem Fehler wird zufällig auf eine der drei Alternativen ersetzt (unabhängig zwischen Positionen). Siehe zur Bernoulli-Verteilung die Übersicht. [5] Die gewählten per-Base-Raten $0.01\,\%$, $0.05\,\%$, $0.10\,\%$ liegen im konservativen Bereich berichteter Illumina-Fehlerraten und bilden eine idealisierte Untergrenze realer Störungen ab. [31]
Das klassische Abweichungsfortpflanzungsgesetz setzt Unabhängigkeit der Quellen voraus. Verletzungen (z. B. Mapping-Abhängigkeiten über Genmodelle) führen zu Unterschätzungen analytischer Formeln und motivieren Monte-Carlo [4]. In unserer Modellierung gelten: (1) unabhängige per-Base-Fehler, (2) nur Substitutionen (keine Indels, keine zyklus-/qualitätsabhängigen Profile), (3) deterministische Pipeline (keine zusätzliche Randomisierung). Abweichungen hiervon (z. B. positionsabhängige Fehler oder zusätzliche stochastische Schritte) erhöhen die beobachtete Streuung und sind Gegenstand künftiger Erweiterungen.

2.2 Formalisierung der Monte-Carlo-Unsicherheitsfortpflanzung

Sei F die Menge der FASTQ-Reads des Datensatzes und $B_{rj} \in \{A, C, G, T\}$ die Base des j-ten Zyklus im Read r. Wir modellieren *aleatorische* Sequenzierfehler als unabhängige Bernoulli-Verteilung [5] mit per-Base-Rate ε und gleichverteilter Substitutionsbasis 1:

$$Z_{rj} \sim \mathrm{Bernoulli}(\varepsilon), \qquad B'_{rj} = \begin{cases} B_{rj}, & Z_{rj} = 0, \\ U_{rj} \sim \mathrm{Uniform}\big(\{A, C, G, T\} \setminus \{B_{rj}\}\big), & Z_{rj} = 1\,. \end{cases}$$

Damit induziert ε ein Produktmaß p_ε auf dem Eingaberaum $\mathcal{F}$ (Menge aller FASTQ-Belegungen). Dieses Modell abstrahiert reale Illumina- Fehlerprofile und wählt bewusst eine *konservative Untergrenze* der Verteilung [31].
Wir modellieren die automatisierte RNA-Seq-Pipeline (Trim Galore! $\rightarrow$ STAR $\rightarrow$ featureCounts $\rightarrow$ DESeq2) als deterministische Abbildung

$$f:\ \mathcal{F} \longrightarrow \mathcal{R},$$

wobei $\mathcal{F}$ den Raum der Eingabedaten (z. B. FASTQ-Dateien) und $\mathcal{R}$ den Raum der Ergebnisobjekte (LFC-Vektoren, p-Werte, DEG-Mengen, GSEA-Resultate etc.) bezeichnet. Die verwendeten Softwarekomponenten sind in den Referenzen dokumentiert. [10, 21, 22] Für ein gegebenes Eingangselement $F' \in \mathcal{F}$ gilt $R = f(F')$. Zur Evaluation der Ausgaben definieren wir vergleichsrelevante Funktionale

$$h:\ \mathcal{R} \longrightarrow \mathbb{R}^q,$$

z. B. $q = 4$ für die Metriken LFC-Kohärenz, Bland-Altman-Bias, Jaccard-Index und GSEA-Kohärenz.
Für eine vorgegebene Fehlerrate ε legen wir eine Verteilungsannahme p_ε über gestörte Eingabedaten fest (z. B. per-Base Substitutionsraten, ggf. mit Korrelationen). Wir ziehen N unabhängige Stichproben

$$F^{(1)}, \ldots, F^{(N)} \overset{\text{iid}}{\sim} p_\varepsilon,$$

propagieren jede Realisation durch die deterministische Pipeline und werten die Funktionale aus:

$$Y^{(i)} \;=\; h\big(f(F^{(i)})\big) \in \mathbb{R}^q, \qquad i = 1, \ldots, N.$$

Als interessierende Kenngrößen betrachten wir z. B. den Erwartungswert

$$\theta(\varepsilon) \;=\; \mathbb{E}_{p_\varepsilon}[\,Y\,]$$

bzw. allgemeine Quantile $Q_\alpha(\varepsilon)$ der Verteilung von Y unter p_ε. Die Monte-Carlo-Schätzer [27] lauten

$$\hat{\theta}_N(\varepsilon) = \frac{1}{N}\sum_{i=1}^{N} Y^{(i)}, \qquad s_Y^2 = \frac{1}{N-1}\sum_{i=1}^{N}\big(Y^{(i)} - \hat{\theta}_N\big)^2,$$

wobei näherungsweise gilt

$$\widehat{\mathrm{Var}}(\hat{\theta}_N) \approx \frac{s_Y^2}{N}, \qquad \mathrm{SE}(\hat{\theta}_N) \approx \frac{s_Y}{\sqrt{N}}.$$

Quantile werden aus den empirischen Ordnungsstatistiken gewonnen (ggf. mit Interpolation).
Beispiele für $h(\cdot)$ in dieser Studie. Als skalare bzw. niedrigdimensionale Bewertungsgrößen h verwenden wir insbesondere:

(i) LFC-Korrelation: $Y_1 = \rho\big(\mathrm{LFC}_{\mathrm{Std}},\, \mathrm{LFC}_{\mathrm{Manip}}\big),$

(ii) Bland-Altman: $d_g = \mathrm{LFC}_g^{\mathrm{Manip}} - \mathrm{LFC}_g^{\mathrm{Std}}, \quad \bar{d} = \frac{1}{G}\sum_g d_g, \quad \mathrm{LoA}_\pm = \bar{d} \pm 1.96\,\mathrm{SD}(d),$

(iii) DEG-Jaccard: $J = \frac{|D_{\mathrm{ref}} \cap D_{\mathrm{test}}|}{|D_{\mathrm{ref}} \cup D_{\mathrm{test}}|},$

(iv) GSEA-Kohärenz: $\kappa_k = \frac{|T_{\mathrm{ref}}^{(k)} \cap T_{\mathrm{test}}^{(k)}|}{k}.$

Die verwendeten Konzepte (Bland-Altman, GSEA, Jaccard) sind etabliert und werden entsprechend zitiert [6, 41, 1].
Analytische Fortpflanzungsformeln (Delta-Methode / Linearisierung) propagieren Momentinformation über ∇f und setzen lokale Linearität sowie bekannte Kovarianzstrukturen voraus [19, 28, 4]. Bei einer mehrstufigen Black-Box-Pipeline mit Nichtlinearitäten, Schwellenoperationen (z. B. FDR-Schwellen) und diskreten Zählgrößen sind diese Voraussetzungen typischerweise verletzt. Die Monte-Carlo-Fortpflanzung schätzt dagegen die durch f induzierten Ausgabeverteilungen direkt und ist unter milden Regularitätsbedingungen konsistent (Gesetz der großen Zahlen, Standardfehler $\sim N^{-1/2}$). Empfehlungen zur Unsicherheitsangabe für komplexe Modelle finden sich in einschlägigen Leitfäden (GUM / NIST) [42, 16].
Per-Base-Fehler werden als unabhängig modelliert (d. h. keine Position-zu-Position-Korrelation) und werden ausschließlich Basensubstitutionen simuliert (keine Indels, keine zyklus- oder qualitätsabhängigen Fehlerprofile). Die Pipeline f wird als deterministisch angenommen (keine zusätzliche Randomisierung/NonDeterminismus). Verstöße gegen diese Annahmen (z. B. positionsabhängige Fehler oder zusätzliche zufällige Komponenten in f) führen zu erhöhter Streuung der $Y^{(i)}$ und sollten in künftigen Erweiterungen explizit modelliert werden.

2.3 Algorithmische Darstellung des Analyse-Workflows

Der methodische Kern dieser Arbeit besteht aus einer Abfolge von vier konzeptionellen Blöcken: der Simulation von Sequenzierfehlern *[1]*, der Ausführung eines Standard-RNA-Seq-Workflows *[2]*, der iterativen Anwendung dieses Workflows zur Untersuchung der Fehlerfortpflanzung *[3]* und der finalen vergleichenden Analyse der Ergebnisse *[4]*. Jeder dieser Blöcke wird im Folgenden als separater Algorithmus dargestellt, um die methodische Vorgehensweise transparent zu machen.

Algorithm 1 Fehlersimulation in FASTQ-Sequenzen

Input: Sequenz S, Fehlerrate ϵ
Output: Modifizierte Sequenz S'

function INTRODUCEERRORS(S, ϵ)
    Initialisiere $S' \leftarrow$ toList(S)
    for $i \leftarrow 0$ **to** length(S) $- 1$ **do**
        if random() $< \epsilon$ **then**
            $b_{original} \leftarrow S'[i]$
            $B_{alphabet} \leftarrow \{'A', 'C', 'G', 'T'\}$
            $b_{new} \leftarrow$ randomChoice($B_{alphabet} \setminus \{b_{original}\}$) ▷ Wähle neue, andere Base
            $S'[i] \leftarrow b_{new}$
        end if
    end for
    return toString(S')
end function

Hinweis zu Algorithmus 1: Dieser Algorithmus beschreibt das Kernstück der Fehlersimulation, die im Python-Skript `introduce_fastq_errors.py` implementiert wurde. Er stellt den stochastischen Prozess dar, bei dem jede Base einer Sequenz mit einer Wahrscheinlichkeit ϵ durch eine andere zufällige Base ersetzt wird.

Algorithm 2 Standard-RNA-Seq Analyse-Workflow

Input: Menge von FASTQ-Dateien F_{in}, Referenzgenom G_{ref}, Genannotation A_{gtf}
Output: DGE-Ergebnistabelle R

function RNASEQPIPELINE(F_{in}, G_{ref}, A_{gtf})
    $F_{trimmed} \leftarrow$ TrimGalore(F_{in}) ▷ Adapter- & Qualitätstrimming
    $F_{aligned} \leftarrow$ STAR($F_{trimmed}, G_{ref}$) ▷ Alignment
    $F_{bam} \leftarrow$ Samtools($F_{aligned}$) ▷ BAM-Prozessierung
    $C \leftarrow$ featureCounts(F_{bam}, A_{gtf}) ▷ Quantifizierung
    $R \leftarrow$ DESeq2Analyse(C) ▷ Differentielle Genexpressionsanalyse
    return R
end function

Hinweis zu Algorithmus 2: Dieser Algorithmus formalisiert die lineare Abfolge der bioinformatischen Werkzeuge, die sowohl für die Referenzanalyse als auch für jede einzelne Simulationsiteration verwendet wird. Er repräsentiert eine Standard-Pipeline zur Analyse differentieller Genexpression und dient als wiederverwendbarer Baustein im Gesamtsystem.

Algorithm 3 Iterativer Workflow der Fehlerfortpflanzungs-Simulation

Input: Originale FASTQ-Dateien F_{orig}, Anzahl Wiederholungen N_{rep}, Fehlerrate ϵ
Output: Menge der Simulationsergebnisse $\{R_1, R_2, ..., R_{N_{rep}}\}$

function RUNERRORSIMULATIONS($F_{orig}, N_{rep}, \epsilon$)
    Initialisiere leere Ergebnismenge $S_{results}$
    for $i \leftarrow 1$ **to** N_{rep} **do**
        $F_{manipulated} \leftarrow$ IntroduceErrorsInFiles(F_{orig}, ϵ) ▷ Siehe Algorithmus 1
        $R_i \leftarrow$ RNASeqPipeline($F_{manipulated}$) ▷ Siehe Algorithmus 2
        Füge R_i zu $S_{results}$ hinzu
        Cleanup($F_{manipulated}$, ...) ▷ Lösche Zwischendateien der Iteration i
    end for
    return $S_{results}$
end function

Hinweis zu Algorithmus 3: Dieser Algorithmus beschreibt die übergeordnete Simulationsschleife, die im Bash-Skript `fehlerfortpflanzung_workflow.sh`[2] implementiert ist. Er verdeutlicht den iterativen Prozess, bei dem für jede von N_{rep} Wiederholungen ein neuer fehlerbehafteter Datensatz generiert und vollständig analysiert wird, bevor die Zwischendateien zur Schonung von Speicherressourcen entfernt werden.

Algorithm 4 Finale vergleichende Analyse

Input: Originale FASTQ-Dateien F_{orig}, Anzahl Wiederholungen N_{rep}, Fehlerrate ϵ
Output: Vergleichs-Plots und -Metriken V

function FINALCOMPARATIVEANALYSIS($F_{orig}, N_{rep}, \epsilon$)
    $R_{std} \leftarrow$ RNASeqPipeline(F_{orig}) ▷ 1. Referenzergebnis generieren
    $S_{results} \leftarrow$ RunErrorSimulations($F_{orig}, N_{rep}, \epsilon$) ▷ 2. Simulationsergebnisse generieren
    Initialisiere leere Menge für Vergleichsmetriken V
    for each R_i **in** $S_{results}$ **do**
        $M_{corr} \leftarrow$ Korrelation(LFC(R_{std}), LFC(R_i)) ▷ LFC-Korrelation
        $M_{bland} \leftarrow$ BlandAltman(LFC(R_{std}), LFC(R_i)) ▷ Bland-Altman-Plot
        $M_{venn} \leftarrow$ VennDiagramm(DEGs(R_{std}), DEGs(R_i)) ▷ DEG-Überschneidung
        $M_{gsea} \leftarrow$ GSEA-Vergleich(Ranks(R_{std}), Ranks(R_i)) ▷ GSEA-Vergleich
        Füge Metriken $\{M_{corr}, M_{bland}, ...\}$ zu V hinzu
    end for
    return V ▷ Aggregierte Ergebnisse und Plots
end function

Hinweis zu Algorithmus 4: Dieser Algorithmus formalisiert die Logik des finalen R-Skripts `fehlerfortpflanzung_vergleich.R`. Er stellt den gesamten wissenschaftlichen Prozess dar: die Generierung der Referenz- und Simulationsdaten, gefolgt von der systematischen, paarweisen Gegenüberstellung der Ergebnisse, um die in der Forschungsfrage formulierte Hypothese zu überprüfen. Für das Alignment der Sequenzdaten wurden das menschliche Referenzgenom `GRCh38.p14` und die zugehörige Genannotation (Ensembl Release 110) [26] als Referenz verwendet.

[2] Aufgrund des Umfangs wurde das vollständige Skript nicht in den Anhang aufgenommen. Das Skript `fehlerfortpflanzung_workflow.sh` ist auf Anfrage verfügbar.

2.4 Studiendesign zur Untersuchung der Fehlerfortpflanzung

Ziel dieses Studiendesigns ist es, gezielt zu messen, wie sich kleine, zufällige Sequenzierfehler durch eine Standard-RNA-Seq-Pipeline auf zentrale Ausgabengrößen fortpflanzen. Konzeptionell bestehen zwei Gruppen: eine *Referenzanalyse* mit unveränderten Originaldaten und eine *Manipulationsanalyse* mit synthetisch gestörten FASTQ-Dateien. Der Gesamtablauf ist in den Algorithmen 1 bis 4 dargestellt. Methodisch handelt es sich um eine *Monte-Carlo-Unsicherheitsfortpflanzung* durch eine deterministische, mehrstufige Pipeline. Die Eingabeunsicherheit wird explizit simuliert und die resultierende Ergebnisunsicherheit empirisch quantifiziert. Eine analytische Fortpflanzung per Delta-Methode ist hier aufgrund von Nichtlinearitäten, Schwellen (FDR) und Tool-Kaskaden wenig verlässlich [19, 28, 16, 42, 4].
Zunächst wird die Pipeline auf den unveränderten Rohdaten ausgeführt und liefert den *Referenzstandard*, an dem alle Simulationen gemessen werden. Die Abfolge (Trim Galore! $\rightarrow$ STAR $\rightarrow$ featureCounts $\rightarrow$ DESeq2) ist in *Algorithmus 2* festgehalten und in den Originalarbeiten dokumentiert [10, 21, 22].
Die Manipulationsanalyse basiert auf einem stochastischen Fehlermodell, dessen Kern in *Algorithmus 1* beschrieben ist. Für jede Base in jeder Sequenz wird mit Wahrscheinlichkeit ε eine Substitution auf eine der drei alternativen Basen vorgenommen. Um eine Dosis-Wirkungs-Perspektive zu erhalten, werden drei Fehlerraten untersucht (*0.01 %*, *0.05 %*, *0.10 %*). Jede fehlerbehaftete Eingabe wird vollständig durch den Standard-Workflow analysiert, der Prozess ist in *Algorithmus 3* skizziert. Das Fehlermodell setzt auf *gleichverteilte Basensubstitutionen* (keine Indels, keine positions- oder qualitätsgewichteten Raten). Es abstrahiert damit reale Illumina-Fehlerprofile und bildet eine konservative Untergrenze der Störung ab, zyklusabhängige Muster oder Phred-gewichtete Varianten können in künftigen Arbeiten ergänzt werden (vgl. Abschnitt 2.1 sowie [31]). Die DGE-Analyse erfolgt mit *DESeq2* (Version 1.48.0) [22] in R (Version 4.5.0). DESeq2 modelliert Rohzählungen mit einer negativen Binomialverteilung, um die typische Überdispersion von RNA-Seq-Daten abzubilden. Das Design lautet `~ donor + condition` und erlaubt die Schätzung des `condition`-Effekts bei gleichzeitiger Kontrolle spenderindividueller Unterschiede (`donor`). Gene mit weniger als 10 Gesamtreads über alle Proben werden vorab entfernt. Als differentiell exprimiert (DEG) gelten Gene mit *padj* < 0.05 (Benjamini-Hochberg [3]) und $|\text{LFC}| > 1$. Die Ergebnisse der Manipulationsanalysen werden systematisch der Referenz gegenübergestellt (vgl. *Algorithmus 4*). Wir werten komplementäre Ebenen aus:

- *Effektstärken (LFC-Robustheit).* Pearson- und Spearman-Korrelationen zwischen Referenz- und Manipulations-LFCs, zusätzlich Bland-Altman-Analysen zur Beurteilung von Bias und Streuung [6].
- *Statistische Signifikanz (DEG-Stabilität).* Überlappung der DEG-Mengen via Venn-Diagramm und Jaccard-Index, globale Effekte auf die Inferenz mittels p-Wert-Verteilungen (Histogramme).
- *Funktionelle Interpretation.* Vergleich der Top-k-Ergebnisse einer Gene Set Enrichment Analysis (GSEA) [41] auf Gene-Ontology-Begriffen [1] unter Verwendung von `clusterProfiler` [44].

3 Realisierung

Dieses Kapitel beschreibt die konkrete technische Umsetzung der im vorherigen Kapitel dargelegten Methodik. Der Fokus liegt auf der Implementierung des automatisierten Workflows, der Struktur der entwickelten Skripte und den spezifischen Parametern der verwendeten Werkzeuge.

3.1 Implementierung des automatisierten Analyse-Workflows

Die gesamte Studie wurde durch ein zentrales Bash-Skript, `fehlerfortpflanzung_workflow.sh`[1], gesteuert. Dieses Skript ermöglicht die Ausführung des gesamten Workflows oder einzelner Teilschritte über Kommandozeilen-Flags und bietet zudem einen interaktiven Menü-Modus. Die Verwaltung der Software-Abhängigkeiten wurde durch eine dedizierte `conda`-Umgebung (`rna-seq-workflow`) sichergestellt, um eine hohe Reproduzierbarkeit zu gewährleisten.

3.1.1 Referenz-Workflow: Verarbeitung der Originaldaten

Der Referenz-Workflow verarbeitet die unveränderten Originaldaten zur Erstellung des Referenzdatensatzes. Er ist im Haupt-Workflow-Skript durch eine Kette von Funktionen implementiert, die nacheinander aufgerufen werden und deren Ablauf in Abbildung 3.1 visualisiert ist.

1. `Datenvorbereitung`: Download und Konvertierung der SRA-Dateien mit dem `SRA Toolkit` (v3.0.0) [38].
2. `Trimming`: Ausführung von `Trim Galore!` (v0.6.7) [18] mit den Parametern `-paired`, `-phred33` und `-length 25`.
3. `Alignment`: Erstellung eines Genom-Index mit `STAR` (v2.7.10a) [10] und `-sjdbOverhang 99`, gefolgt vom Alignment der Reads.
4. `BAM-Prozessierung`: Konvertierung, Sortierung und Indexierung mittels `samtools` (v1.15.1) [20].
5. `Quantifizierung`: Erstellung der Zählungsmatrix mit `featureCounts` (v2.0.3) [21] unter Verwendung der Parameter `-p` (paired-end) und `-s 0` (unstranded).

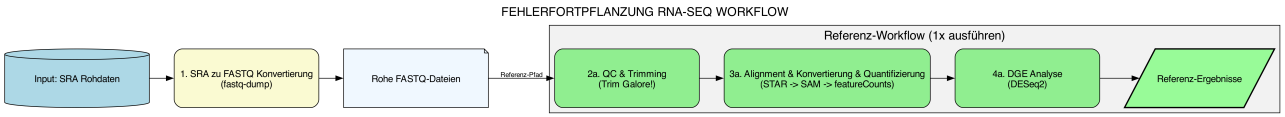

Abbildung 3.1: Schematische Darstellung des Referenz-Workflows zur Prozessierung der Originaldaten. Jeder Kasten repräsentiert einen Prozessschritt, der durch ein spezifisches bioinformatisches Werkzeug realisiert wird.

[1] Aufgrund des Umfangs wurde das vollständige Skript nicht in den Anhang aufgenommen. Das Skript `fehlerfortpflanzung_workflow.sh` ist auf Anfrage verfügbar.

3.1.2 Fehlerfortpflanzungs-Workflow: Iterative Simulation

Der Fehler-Workflow, dessen Ablauf in Abbildung 3.2 dargestellt ist, wird durch die Funktion `run_error_propagation_workflow` im Bash-Skript gesteuert. Diese Funktion implementiert eine äußere Schleife, die über die definierte Anzahl an Wiederholungen (`$N_REPETITIONS`) iteriert.
Bei der Fehlersimulation wurde in jeder Iteration zunächst ein Satz fehlerbehafteter FASTQ-Dateien mit dem Python-Skript `introduce_fastq_errors.py` erzeugt. Das Skript nimmt die Original-FASTQ-Dateien und die gewünschte Fehlerrate (z.B. `-error_rate 0.05`) als Input und generiert neue, komprimierte FASTQ-Dateien.
Die neu erzeugten fehlerbehafteten FASTQ-Dateien durchlaufen anschließend die gleichen Prozessschritte (Trimmen, Alignment, BAM-Konvertierung) wie im Referenz-Workflow (Pipeline-Ausführung pro Iteration). Um die Kompatibilität mit nachfolgenden Analyseschritten zu gewährleisten und den Prozess für Cluster-Umgebungen zu parallelisieren, wurde eine spezielle Logik für die Quantifizierung implementiert:

1. Die für eine Iteration erzeugten BAM-Dateien werden temporär in das zentrale Verzeichnis `bam_trimmed/` kopiert.

2. `featureCounts` wird auf den gesamten Inhalt dieses Verzeichnisses angewendet, um eine einzelne Zählungsmatrix für diese Iteration zu erstellen.

3. Unmittelbar nach erfolgreicher Quantifizierung werden alle Zwischendateien dieser Iteration, einschließlich der temporär kopierten BAMs, automatisch gelöscht, um den Speicherplatz effizient zu nutzen.

Dieser Prozess wird für jede Wiederholung wiederholt, was zu einer Serie von Zählungsmatrizen im Verzeichnis `counts/counts_error/` führt.

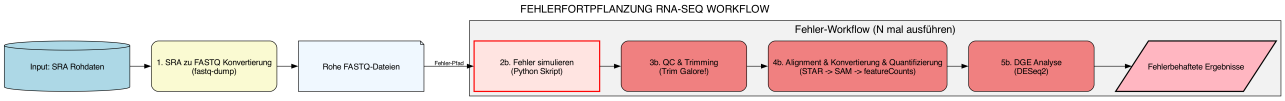

Abbildung 3.2: Schematische Darstellung des Fehlerfortpflanzungs-Workflows. Der Prozess beginnt mit der Einführung von Fehlern in die Originaldaten und durchläuft anschließend dieselbe Analysepipeline wie der Referenz-Workflow.

Für jede Probe wurden Reads/Probe, Mapping-Raten (uniquely/multi-mapped), Duplikate, Detektionsrate (Anteil Gene mit CPM $>$ 1 in $\geq$ 50 % der Proben), Insert-Länge und Q30 ermittelt siehe Anhang, Tabelle 1 (SRA-Metadaten).
Die folgende Darstellung skizziert kurz das Wirkprinzip des verwendeten Algorithmus. Die Parallelisierung ist nicht Gegenstand dieser Arbeit. Das hier eingesetzte Cluster-Muster dient ausschließlich dazu, die Ausführung effizienter zu gestalten [23, 24].

Algorithm 5 Kritischer Abschnitt mit Lock, Trap und Zufallswartezeit

```
1:  procedure WithExclusiveLock(lock_dir, action())
2:      SetTrap(INT, TERM, EXIT)                    ▷ RemoveDir(), LogError(), Exit(1)
3:      while not MakeDir(lock_dir) do
4:          sleep_time ← ∈ {5, ..., 14}             ▷ RANDOM % 10 + 5
5:          LogWarn("gesperrt ← sleep_time")
6:          Sleep(sleep_time)
7:      end while
8:      LogInfo("Lock erhalten")
9:      rc ← action()                    ▷ kritischer Abschnitt (z. B. featureCounts)
10:     RemoveDir(lock_dir)                                  ▷ Lock freigeben
11:     ClearTrap()
12:     return rc
13: end procedure
```

Der Algorithmus erzwingt exklusiven Zugriff über ein Lock-Verzeichnis (`mkdir` atomar). Solange das Lock belegt ist, wartet er mit einer zufälligen Zeit $5 + (\$RANDOM \bmod 10)$ Sekunden, um Kollisionen zu entzerren. Mit `trap` werden bei `INT`, `TERM` und `EXIT` Aufräumaktionen (Lock löschen, Logging) garantiert. Nach Ausführung wird das Lock freigegeben. Vgl. Bash-Handbuch: *Signals* (trap) und *Special Parameters* ($RANDOM) [33].

3.2 Implementierung der finalen Vergleichsanalyse

Die abschließende Auswertung und der Vergleich aller Ergebnisse wurden in einem einzigen, umfassenden R-Skript, `fehlerfortpflanzung_vergleich.R`, realisiert. Dieses Skript wurde entwickelt, um den gesamten Analyse- und Visualisierungsprozess in einem Durchlauf durchzuführen.
Das Skript definiert eine zentrale Funktion,
`run_deseq_analysis()`, die eine vollständige DGE-Analyse für eine gegebene Zählungsmatrix durchführt. Diese Funktion wird zunächst für die Referenz-Zählungsmatrix und anschließend für eine exemplarische fehlerbehaftete Zählungsmatrix aufgerufen. Sie führt nicht nur die `DESeq2`-Analyse durch, sondern generiert auch alle diagnostischen Plots (MA, Volcano, PCA, Heatmap) und speichert die Ergebnisse in dedizierten Ordnern (`output/standard/` bzw. `output/manipulated/`).
Nach der Ausführung der beiden Einzelanalysen lädt das Skript die jeweiligen Ergebnistabellen und führt die in der Methodik beschriebenen Vergleiche durch. Die Implementierung nutzt intensiv Pakete des `tidyverse` für die Datenaggregation und `ggplot2` für die Erstellung der Vergleichsplots (Korrelationsplot, Bland-Altman-Plot, p-Wert-Histogramme). Für das Venn-Diagramm wurde das Paket `ggvenn` verwendet. Alle finalen Vergleichsplots sowie die diagnostischen Plots beider Analysen werden sowohl als einzelne PNG-Dateien als auch in einem aggregierten PDF-Bericht (`final_summary_report.pdf`) im Verzeichnis `output_final/` gespeichert.

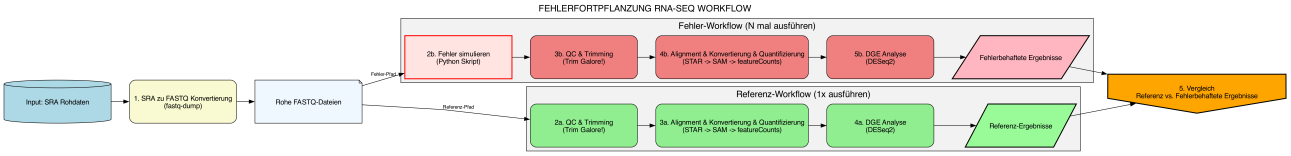

Abbildung 3.3: Gesamtübersicht des automatisierten RNA-Seq-Workflows mit zwei Pfaden.

Die Abbildung fasst die beiden zuvor getrennt gezeigten Abläufe (Abb. 3.1 und Abb. 3.2) in einer Gesamtansicht zusammen. Der Standard-Workflow erzeugt den Referenzdatensatz, der Fehler-Workflow

wiederholt die identische Pipeline auf künstlich gestörten FASTQs. So werden die Effekte kleiner Sequenzierfehler über den gesamten Prozess bis hin zu DGE/GSEA sichtbar und mit konsistenten Metriken quantifiziert. *Pfeile unten:* Referenz-Workflow von den Rohdaten (SRA/FASTQ) bis zur DGE/GSEA-Auswertung. *Pfeile oben:* Fehlerfortpflanzungs-Workflow, in dem vor der Pipeline definierte Basensubstitutionen mit Raten $\varepsilon \in \{0.01\,\%, 0.05\,\%, 0.10\,\%\}$ injiziert werden und die gleiche Verarbeitung pro Iteration erfolgt ($N = 1000$). Gemeinsame Schritte: Trimming (`Trim Galore!`), Alignment (`STAR`), BAM-Prozessierung (`samtools`), Quantifizierung (`featureCounts`), DGE (`DESeq2`). Die Auswertung vergleicht je Iteration Referenz vs. Manipulation anhand von LFC-Korrelation, Bland-Altman-Bias/LoA, Jaccard-Index und GSEA-Kohärenz.

4 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Analysen vorgestellt. Im Mittelpunkt steht der systematische Vergleich zwischen der fehlerfreien Referenzanalyse *Standard* und Analysen mit künstlich eingeführten Fehlerraten von 0.01 %, 0.05 % und 0.10 % (*Manipuliert*). Ziel ist es, die Auswirkungen der Fehlerfortpflanzung auf unterschiedlichen Ebenen der Datenauswertung zu quantifizieren von globalen Mustern bis hin zur biologischen Interpretation.

4.1 Hohe Robustheit globaler Expressionsmuster und Effektstärken

Die erste zentrale Beobachtung ist die hohe Stabilität der globalen Datenstruktur und der geschätzten Genexpressionsänderungen, selbst bei erhöhten Fehlerraten.

4.1.1 Konsistenz der Proben-Clusterung in der Hauptkomponentenanalyse

Die Principal Component Analysis (PCA) wurde verwendet, um die globale Ähnlichkeit zwischen Proben zu visualisieren und primäre Varianzquellen zu identifizieren. Die PCA transformiert dieses hochdimensionale System in Hauptkomponenten, um die maximale Varianz zu erklären. Abbildung 4.1 vergleicht die PCA-Plots der Referenzanalyse mit denen der Analysen mit 0.01 %, 0.05 % und 0.10 % simulierten Sequenzierfehlern.

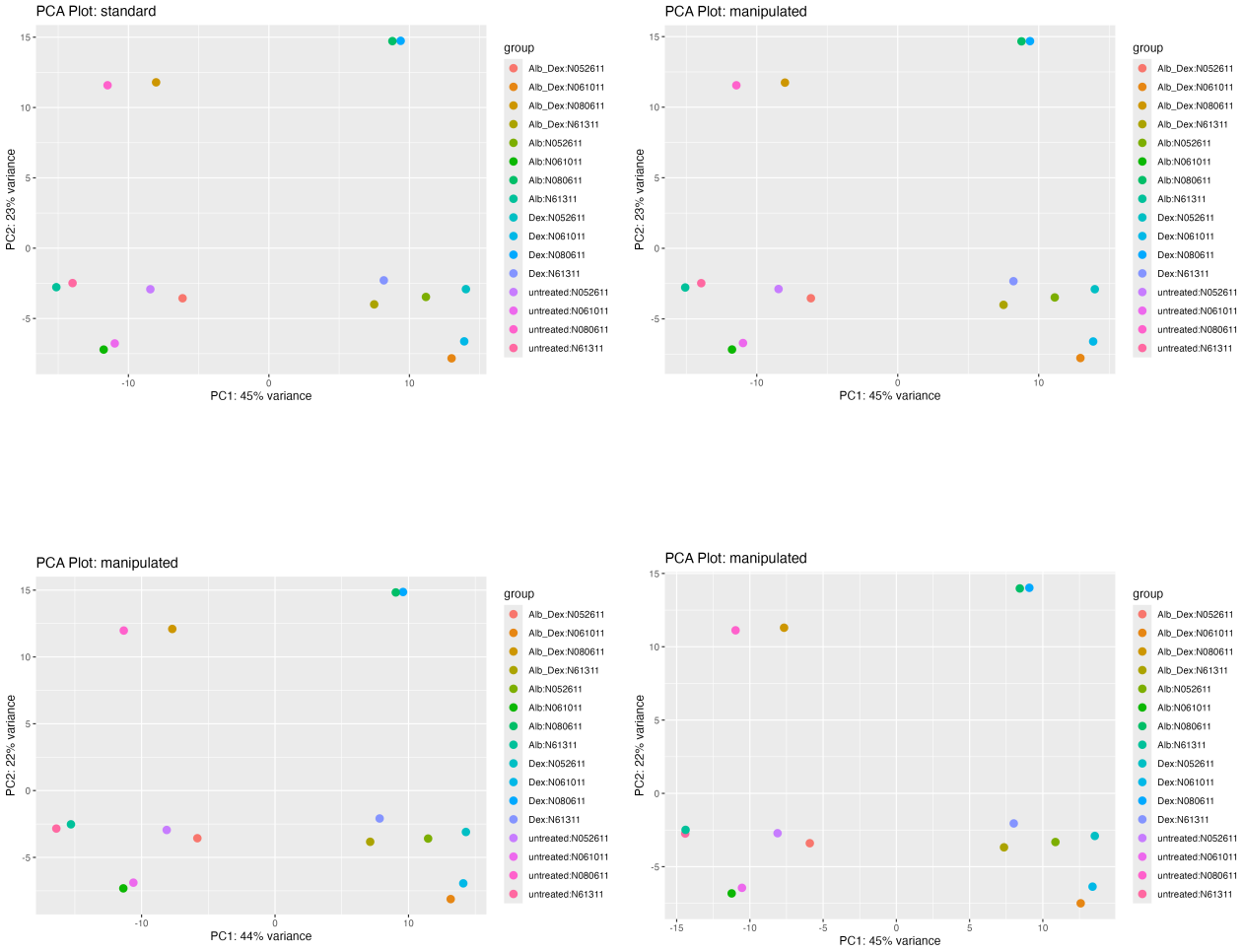

Abbildung 4.1: Vergleich der PCA-Plots (oben links *Standard*, oben rechts 0.01 %, unten links 0.05 %, unten rechts 0.10 %). Die Achsentitel geben den jeweiligen Varianzanteil an. Die Proben-Clusterung nach Spender (`donor`) bleibt in allen Szenarien dominant, die Behandlung trennt sekundär.

Die Abbildung zeigt eine hohe Konsistenz der globalen Probenverteilung. In allen vier Szenarien gruppieren sich die Proben klar nach dem Spender. Die interindividuelle Variabilität ist damit die stärkste biologische Varianzquelle. Das durch Fehler induzierte Rauschen überdeckt diese primären Signale nicht und verändert die Clusterstruktur nicht wesentlich.

4.1.2 Stabilität der $\log_2$-Fold-Changes

Neben den globalen Mustern wurde die Robustheit der auf Genebene geschätzten Effektstärken (LFCs) untersucht. Der Vergleich der LFC-Werte zwischen Referenz und manipulierten Datensätzen zeigt eine sehr hohe Übereinstimmung (Abb. 4.2).

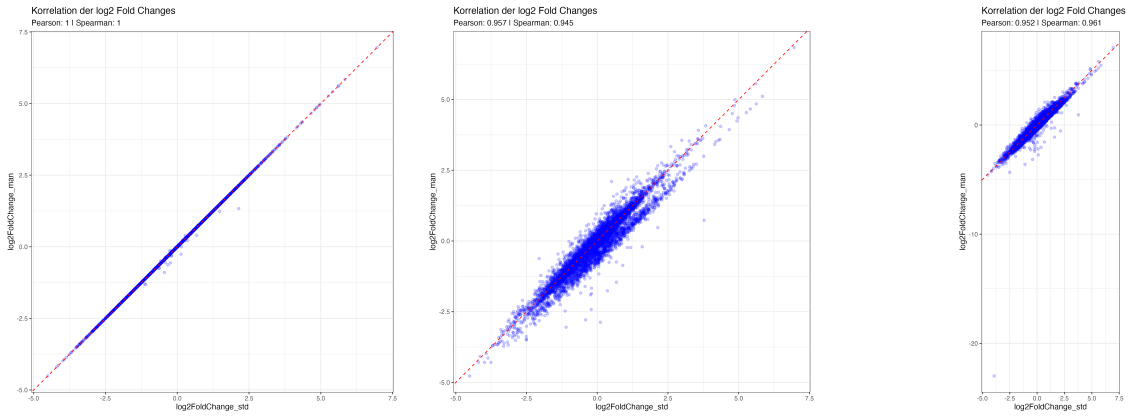

Abbildung 4.2: Vergleich der $\log_2$-Fold-Changes zwischen Referenz und Simulation (Scatter/Korrelation, links 0.01 %, mitte 0.05 %, rechts 0.10 %). Durchgezogene Linie: Identitätslinie, Werte nahe der Linie zeigen hohe Übereinstimmung.

Selbst bei 0.10 % (rechts) liegt der Pearson-Korrelationskoeffizient nahe 1. Dies deutet darauf hin, dass die Methode zur Berechnung der differentiellen Expression unempfindlich gegenüber eingeführtem Zufallsrauschen ist. In den Streudiagrammen sind zwei Punktwolken erkennbar, die unterschiedliche Genpopulationen (z. B. niedrige vs. hohe Abundanz) widerspiegeln. Beide behalten eine starke lineare Beziehung, was die Robustheit der LFC-Schätzung für beide Gruppen unterstreicht.
Über $N = 1000$ bleibt die mediane LFC-Korrelation (Pearson/Spearman) in allen Fehlerraten hoch, die mediane Breite der Limits of Agreement (LoA) nimmt moderat zu.

4.1.3 Bland–Altman-Plots: Bias und Limits of Agreement

Die Bland–Altman-Analyse [6] bewertet die *Übereinstimmung* der LFC-Schätzungen zwischen Referenz und Manipulation, indem für jedes Gene $d_g = \mathrm{LFC}_g^{\mathrm{Manip}} - \mathrm{LFC}_g^{\mathrm{Std}}$ gegen $m_g = \frac{1}{2}(\mathrm{LFC}_g^{\mathrm{Manip}} + \mathrm{LFC}_g^{\mathrm{Std}})$ geplottet wird. Die durchgezogene Linie zeigt den mittleren Unterschied (Bias $\bar{d}$), die gestrichelten Linien die 95 %-*Limits of Agreement* $\bar{d} \pm 1.96\,\mathrm{SD}(d)$.

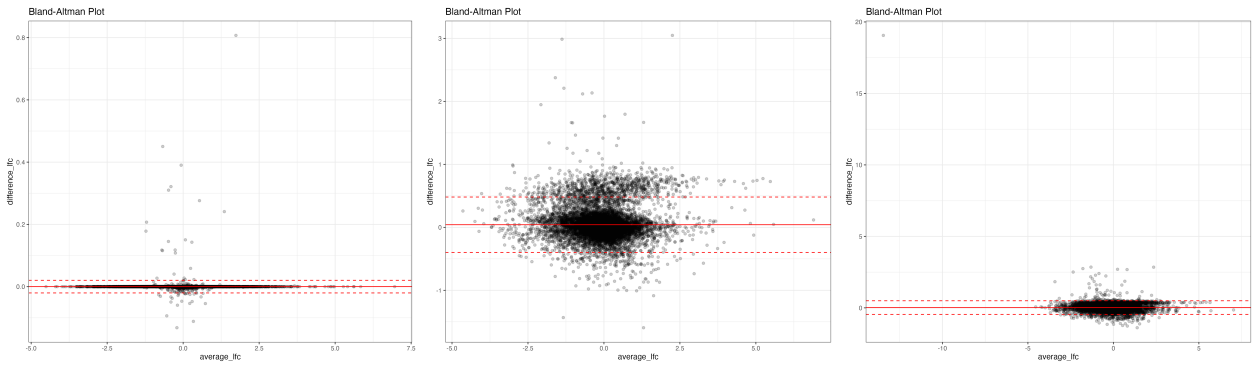

Abbildung 4.3: Bland–Altman-Vergleiche der $\log_2$-Fold-Changes (*Standard* vs. ε): *links* 0.01 %, *mitte* 0.05 %, *rechts* 0.10 %. Durchgezogene Linie: Bias, gestrichelte Linien: LoA $\bar{d} \pm 1.96\,\mathrm{SD}(d)$.

In allen Vergleichen liegt $\bar{d}$ nahe 0, es gibt also keine systematische Verschiebung zugunsten einer Bedingung. Mit steigender Fehlerrate weiten sich die LoA sichtbar auf. Die Streuung der LFC-Differenzen steigt damit vorhersagbar mit dem Eingaberauschen, die zentrale Tendenz bleibt stabil. Bei kleinen bis mittleren Mittelwerten m_g (oft geringere Abundanz) ist die Streuung größer als bei hohen m_g konsistent mit den MA-Plots. Es zeigt sich keine konsistente Neigung von d_g über m_g (kein proportionaler Bias). Optional lässt sich dies per Regression $d_g \sim m_g$ prüfen. Effektstärkere Gene bleiben stabil (enge

Differenzen), Rauschen verbreitert die Übereinstimmungsgrenzen vor allem bei geringabundanten Genen. Neben Korrelationen sollten daher stets Bias und LoA berichtet werden (LoA-Breite als Stabilitätskennzahl).

4.1.4 MA-Plots: Abundanzabhängigkeit der LFC-Streuung

Der MA-Plot stellt für jedes Gen die Effektstärke ($M = \log_2$-Fold-Change) gegen die mittlere Abundanz (A, z. B. $\log_{10}$ der normalisierten Counts) dar und markiert signifikante Gene (padj < 0.05). In allen vier Szenarien (Standard sowie Manipulationen mit $\varepsilon \in \{0.01\,\%, 0.05\,\%, 0.10\,\%\}$) zeigen die MA-Plots ein konsistentes Gesamtbild (vgl. [22]):
Die Trendkurve (LOESS) liegt über weite Bereiche von A nahe $M = 0$ im Einklang mit Bias ≈ 0 und hohen LFC-Korrelationen. Die typische Trichterstreuung bei kleinen A ist im Referenzfall sichtbar und nimmt mit wachsender Fehlerrate moderat zu. Hochabundante Gene behalten weitgehend stabile LFCs. Mit steigender ε wechseln Grenzfälle häufiger die Seite der FDR-Schwelle (Flip-Events). Die Anzahl signifikanter Punkte ändert sich nur moderat, die Identität wechselt konsistent mit sinkendem Jaccard-Index. Gene mit großen $|M|$ und hoher Abundanz bleiben über alle Bedingungen hinweg nahezu unverändert. Die in DESeq2 verwendete LFC-Shrinkage dämpft Extrema insbesondere bei kleinen A [22]. Dadurch bleibt die Zunahme der Streuung unter Rauschen kontrolliert. Effektstärken sind global stabil und weitgehend unbeeinflusst von kleinen Sequenzierfehlern, verwundbar ist die binäre Signifikanzentscheidung bei geringabundanten Genen. Berichte sollten dies transparent machen (Trendkurve, Grenzregion).

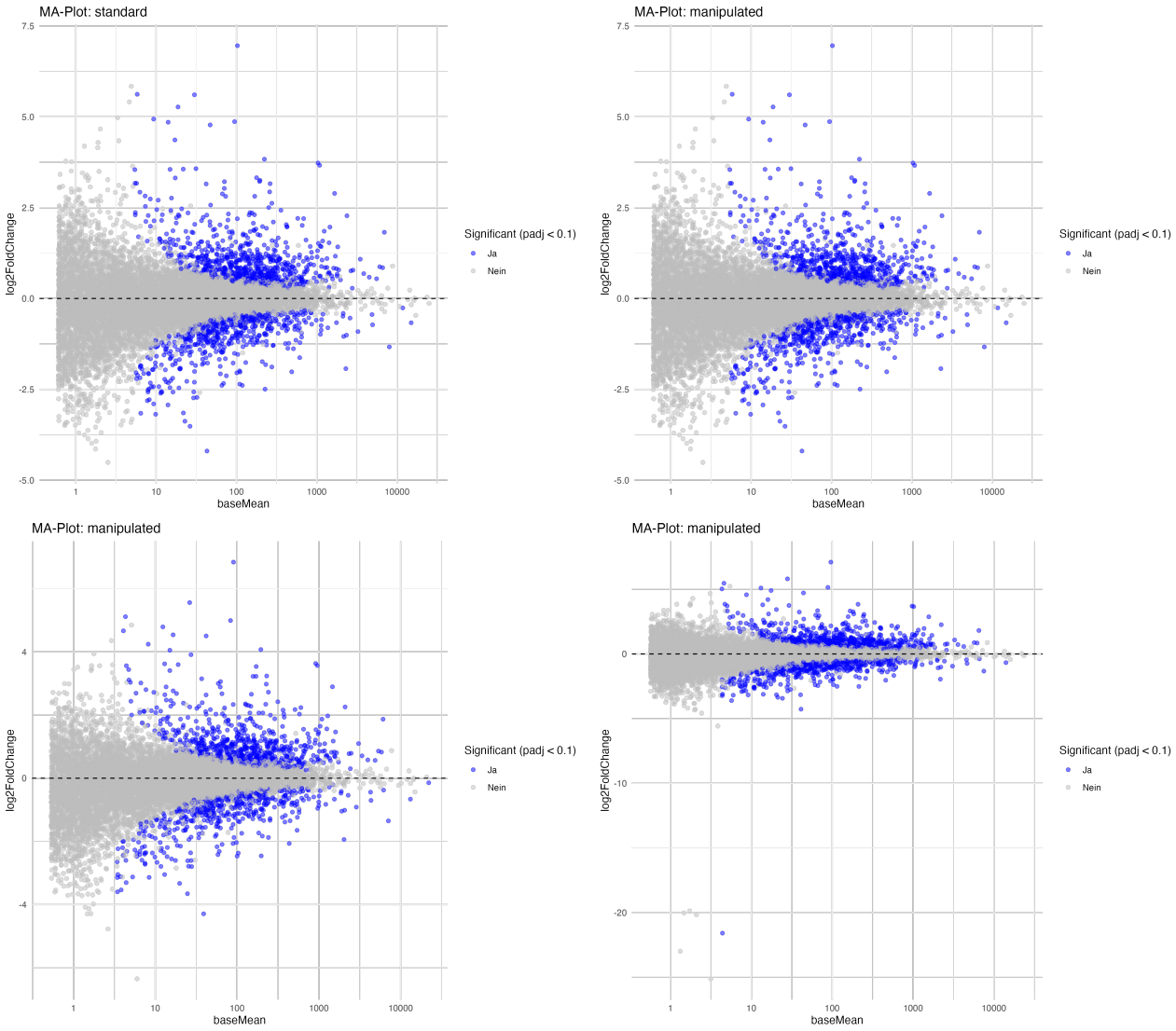

Abbildung 4.4: MA-Plots für *Standard* (oben links) und Manipulationen mit $\varepsilon \in \{0.01\,\%, 0.05\,\%, 0.10\,\%\}$. Signifikante Punkte (padj < 0.05) sind hervorgehoben, die LOESS-Trendlinie verläuft nahe $M = 0$.

4.2 Messbare Instabilität an der Schwelle der statistischen Signifikanz

Im Gegensatz zur Robustheit der globalen Metriken zeigen sich bei den statistischen Signifikanzen deutliche, mit der Fehlerrate zunehmende Instabilitäten.

4.2.1 Volcano-Plots: Signifikanz vs. Effektstärke unter Rauschen

Der Volcano-Plot stellt Effektstärke ($x = \log_2$-Fold-Change) und Evidenz ($y = -\log_{10}(\text{padj})$) gegenüber. Vertikale Linien markieren den LFC-Schwellenwert ($|\,\text{LFC}| > 1$), die horizontale Linie die FDR-Grenze (padj $= 0.05$). Abbildung 4.5 zeigt den Vergleich für *Standard* und die drei Fehlerraten.

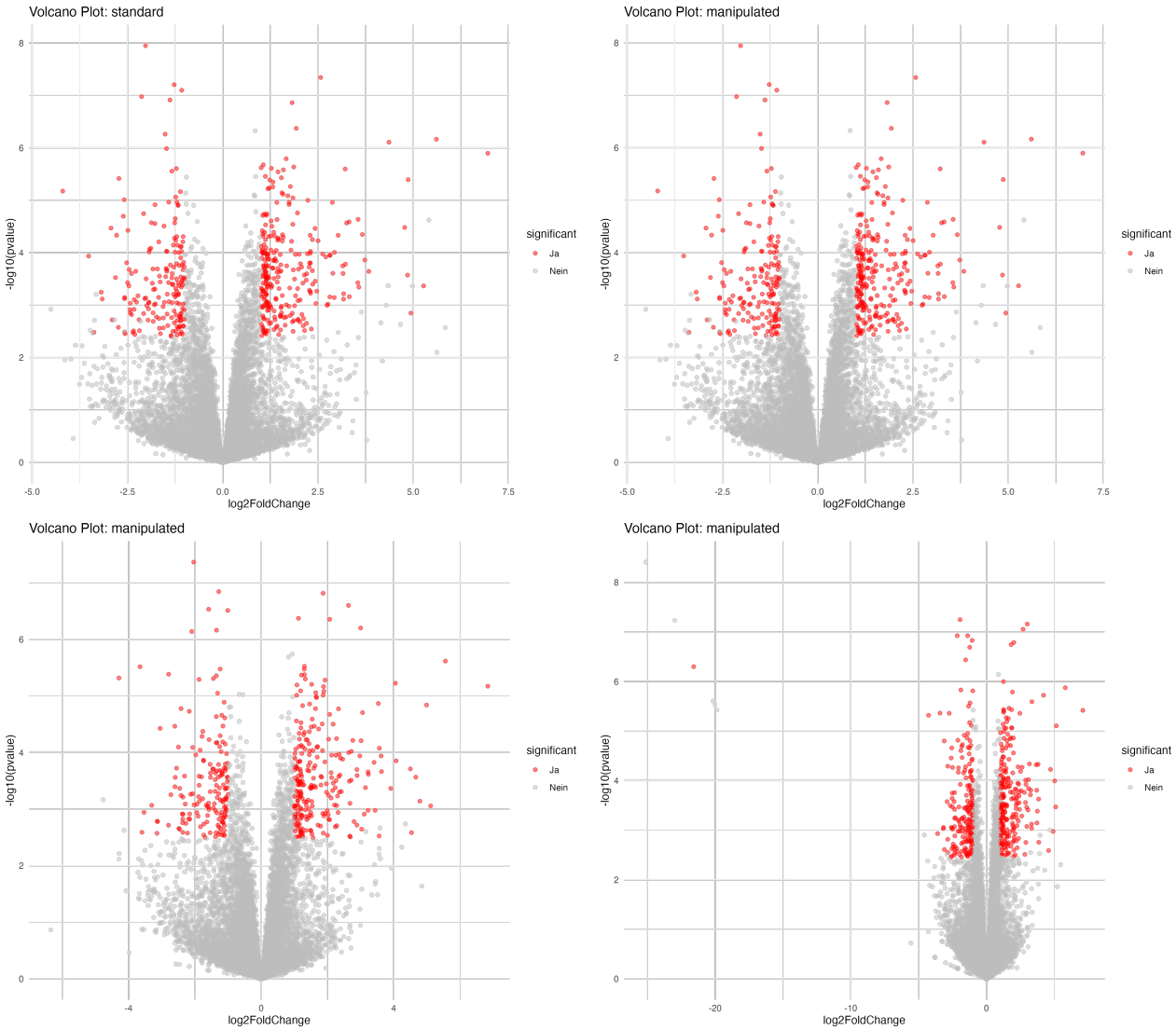

Abbildung 4.5: Volcano-Plots für *Standard* (oben links) und Manipulationen mit $\varepsilon \in \{0.01\,\%, 0.05\,\%, 0.10\,\%\}$ (oben rechts, unten links, unten rechts). x-Achse: LFC, y-Achse: $-\log_{10}(\text{padj})$. Horizontale Linie: FDR = 0.05, vertikale Linien: $|\text{LFC}| = 1$.

Die Punktewolken bleiben weitgehend symmetrisch um LFC$= 0$ im Einklang mit Bias ≈ 0 und hohen LFC-Korrelationen. Mit steigender Fehlerrate nimmt die Dichte oberhalb der FDR-Grenze etwas ab, besonders bei moderaten Effekten ($|\text{LFC}| \approx 1 \ldots 1.5$). Ursache ist Varianz-Inflation bei niedriger/mittlerer Abundanz (vgl. Abs. 4.1.4). Nahe der FDR-Linie treten Flip-Events häufiger auf: Grenzfälle fallen unter die Linie (potenziell falsch-negativ), andere erscheinen knapp darüber (potenziell falsch-positiv). Die Anzahl signifikanter Gene ändert sich moderat, die Identität wechselt konsistent mit dem sinkenden Jaccard-Index (vgl. Abb. 4.7). Gene mit großen Effektstärken und guter Abdeckung bleiben deutlich über der FDR-Linie (vgl. Abb. 4.8).

4.2.2 p-Wert-Verteilungen: Kalibrierung und Erosion der Signale

Die Verteilungen der *ungekorrigierten* p-Werte (Wald-Test in DESeq2) ergänzen die Volcano-Befunde und zeigen die Wirkung zufälliger Sequenzierfehler auf die globale Evidenzverteilung.

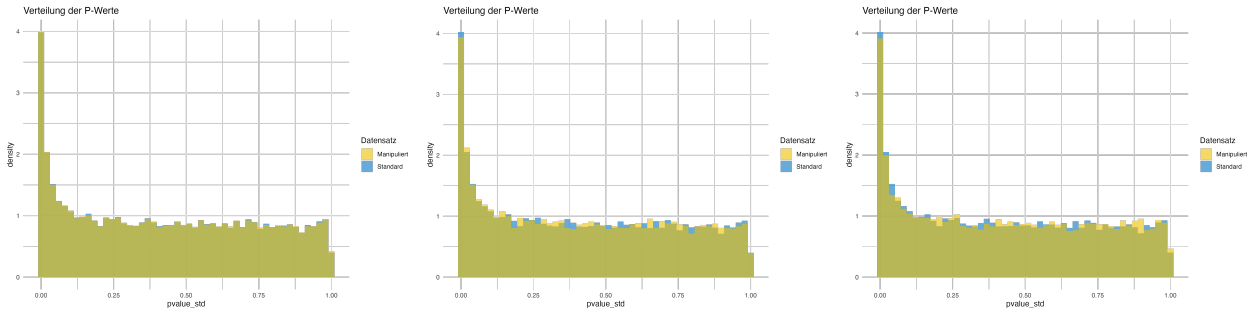

Abbildung 4.6: Histogramme der unadjustierten p-Werte (Wald-Test, DESeq2) in den drei Manipulationsbedingungen: *links* $\varepsilon = 0.01\,\%$, *mitte* $\varepsilon = 0.05\,\%$, *rechts* $\varepsilon = 0.10\,\%$. Eine gut kalibrierte Null erzeugt annähernd uniforme Verteilungen, echte Effekte erzeugen einen Ausschlag nahe 0. Mit steigender Fehlerrate nimmt die Dichte kleiner p-Werte ab (Signal-Erosion), während mittlere p-Werte zunehmen.

Bei $\varepsilon = 0.01\,\%$ zeigt sich noch ein dem Referenzfall ähnliches Muster: nahezu uniforme Grundverteilung mit Ausschlag nahe $p \approx 0$. Von 0.01 % zu 0.10 % verlagert sich Masse vom linken Rand $[0, 0.05)$ in mittlere Bereiche (~ 0.1 bis 0.5). Ursache ist Varianz-Inflation/Effektdämpfung bei geringer bis mittlerer Abundanz: Teststatistiken werden kleiner, p-Werte steigen. Das ist konsistent mit den Volcano- und MA-Befunden. Es gibt keine ungewöhnliche Häufung bei $p \approx 1$ oder starke Multimodalität. Die Verschiebung entspricht einem Powerverlust, nicht einer grundsätzlichen Fehlkalibrierung des Tests. Benjamini–Hochberg [3] weist weniger Gene als signifikant aus, wenn die Gesamtheit der p-Werte weniger stark linkslastig ist (effektiv höherer π_0-Anteil). Das erklärt den rückläufigen DEG-Umfang und den sinkenden Jaccard-Index.

4.2.3 Divergenz der Listen differentiell exprimierter Gene

Die Venn-Diagramme in Abbildung 4.7 visualisieren die Übereinstimmung der als signifikant klassifizierten Gene (*padj* < 0.05) und offenbaren eine klare Dosis-Wirkungs-Beziehung zwischen Fehlerrate und Ergebnisstabilität.

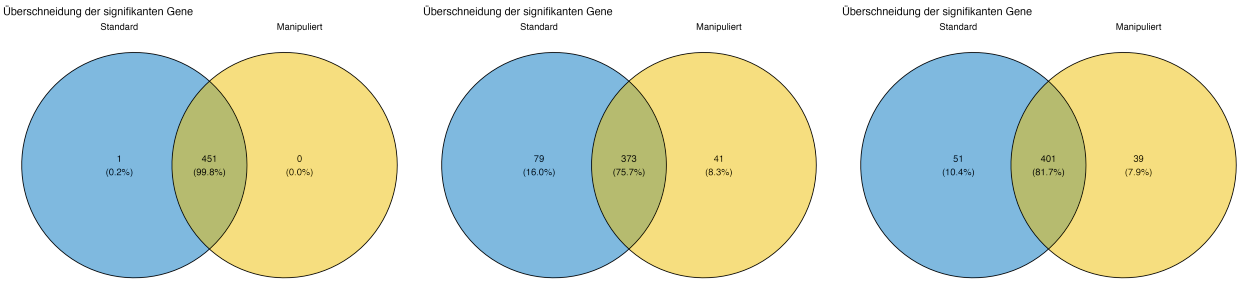

Abbildung 4.7: Überlappung der DEG-Listen (padj < 0.05) zwischen Referenz und Simulation. Der Jaccard-Index quantifiziert die Schnittmenge relativ zur Vereinigung. *Links* $\varepsilon = 0.01\,\%$, *mitte* $\varepsilon = 0.05\,\%$, *rechts* $\varepsilon = 0.10\,\%$.

Bei einer Fehlerrate von 0.01 % ist die Übereinstimmung mit der Referenzanalyse noch nahezu perfekt, die Listen sind fast identisch. Mit steigender Fehlerrate wächst die Divergenz jedoch deutlich an. Bei 0.05 % teilen die Listen nur noch rund 76 % der Gene, während bei 0.10 % die Zahl der nicht-gemeinsamen Gene weiter zunimmt. Dieser Trend wird durch den Jaccard-Index das Verhältnis

der Schnittmenge zur Vereinigungsmenge quantifiziert, der mit zunehmender Fehlerlast tendenziell sinkt. Diese zunehmende Diskrepanz ist eine direkte Konsequenz der in den Volcano- und MA-Plots beobachteten *Schwellenfragilität*. Die durch das Rauschen verursachte Varianz-Inflation führt dazu, dass p-Werte von Genen nahe der Signifikanzgrenze schwanken. Gene verlieren dadurch ihre Signifikanz oder sie gewinnen neu hinzu interpretierbar. Die Zahl dieser sogenannten *Flip-Events* steigt mit der Fehlerlast, was die Listen der differentiell exprimierten Gene (DEGs) zunehmend instabil macht. Die Analyse bestätigt, dass die binäre Entscheidung *signifikant* vs. *nicht signifikant* hochgradig von der initialen Datenqualität abhängt. Eine reine DEG-Liste ist daher für sich genommen nur begrenzt aussagekräftig. Ihre Stabilität sollte immer kritisch hinterfragt und idealerweise durch quantitative Maße wie den Jaccard-Index gestützt werden.

4.2.4 Veränderung der Top-DEGs

Die Robustheit der globalen Muster bedeutet nicht, dass *identische* Gene in den Spitzenrängen verbleiben. Abbildung 4.8 zeigt die *Top 30 DEGs* jeder Analyse (Auswahl nach *padj*, bei Gleichstand nach |LFC|) als Heatmaps: einmal für die Referenz und jeweils für $\varepsilon \in \{0.01\,\%, 0.05\,\%, 0.10\,\%\}$.

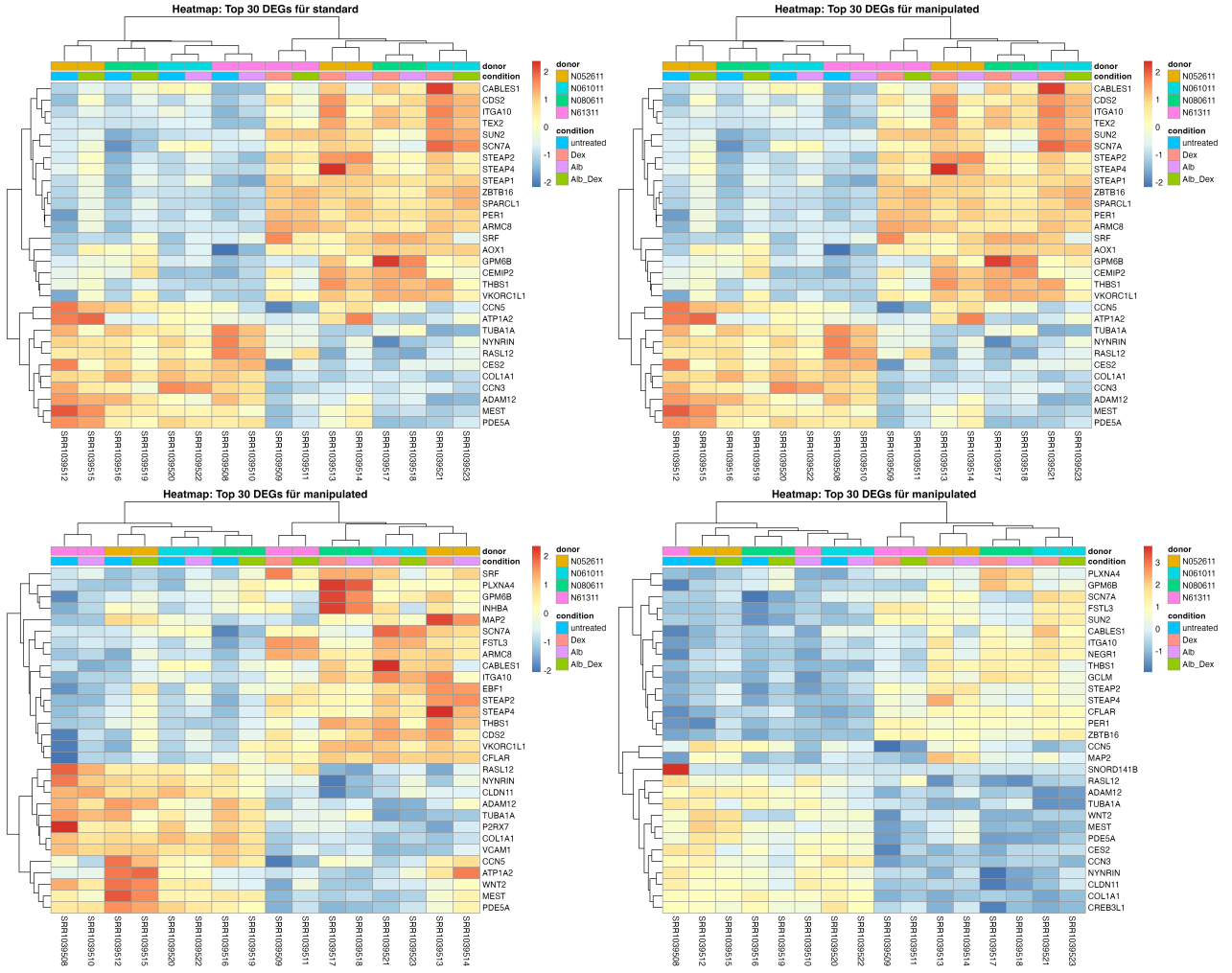

Abbildung 4.8: Top-30-DEG-Heatmaps pro Analyse: *oben links* Referenz, *oben rechts* $\varepsilon = 0.01\,\%$, *unten links* $\varepsilon = 0.05\,\%$, *unten rechts* $\varepsilon = 0.10\,\%$. Gezeigt ist je Analyse die nach *padj* (Tiebreaker |LFC|) ausgewählte Top-30-Liste, z-skaliert pro Gen (Zeile). Die Proben clustern in allen Szenarien ähnlich, auffällig ist die wechselnde Identität einzelner Top-Hits mit steigender Fehlerrate.

Die hierarchische Clusterung reproduziert die in der PCA beobachtete Struktur: donor-getriebene Gruppenbildung bleibt erhalten, Behandlungseffekte trennen sekundär. Mit wachsender Fehlerrate nimmt die Überschneidung der Top-30-Listen mit der Referenz sichtbar ab insbesondere bei Genen mit geringer/mittlerer Abundanz und moderatem Effekt (Grenzfälle an der FDR-Schwelle). Gene mit großen Effektstärken und guter Abdeckung bleiben überwiegend in den Top-Rängen präsent. Das erklärt die Robustheit globaler Effektstärke-Metriken trotz wechselnder Top-30-Zusammensetzung.

4.2.5 Variabilität in der Gene Set Enrichment Analysis

Die GSEA [41] auf Gene Ontology (GO) [1] wurde jeweils *paarweise* zwischen der Referenzanalyse und den manipulierten Datensätzen verglichen. Abbildung 4.9 zeigt die drei Vergleiche (Standard vs. $\varepsilon \in \{0.01\,\%, 0.05\,\%, 0.10\,\%\}$), jeweils als Gegenüberstellung der ranghöchsten GO-Terme (z. B. nach NES/FDR), berechnet mit `clusterProfiler` [44].

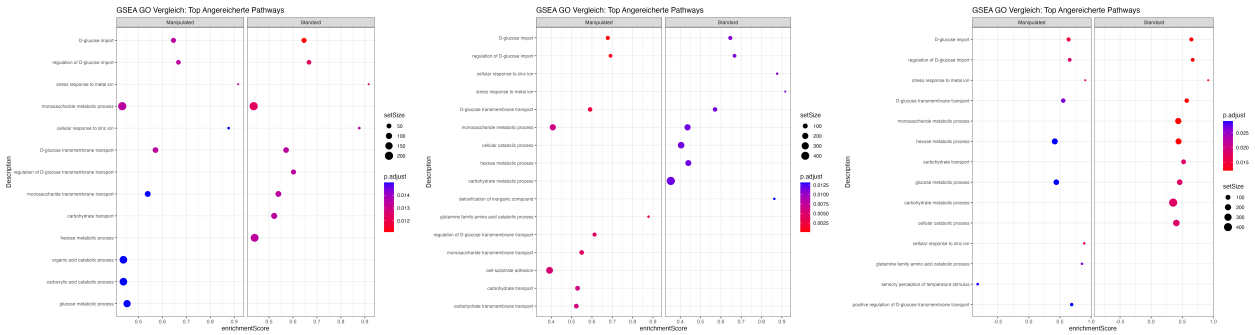

Abbildung 4.9: Paarweise GSEA-Vergleiche *Standard vs.* ε: *links* 0.01 %, *mitte* 0.05 %, *rechts* 0.10 %. Übergeordnete Themen bleiben sichtbar, während sich Rangfolge und FDR einzelner Terme mit steigender Fehlerrate verschieben.

Die GSEA-Vergleiche zeigen, dass die übergeordneten biologischen Programme im Wesentlichen erhalten bleiben. Das passt zur Stabilität der globalen Muster und der LFC-Schätzungen. Veränderungen treten vor allem auf Ebene einzelner GO-Terme auf: Mit steigender Fehlerrate verschieben sich NES und FDR bei Begriffen mit moderaten Effekten oder kleiner Leading-Edge, wodurch Terme die Top-Liste verlassen oder neu hinzukommen. Diese Rangdrift spiegelt die in den MA-Plots beobachtete, abundanzabhängige Streuungszunahme wider (Abs. 4.1.4). Begriffe, deren Leading-Edge überwiegend aus geringabundanten Genen besteht, reagieren empfindlich. Sets, die von hochabundanten Genen mit großen Effekten getragen werden, bleiben stabil. Für die biologische Auswertung ist daher weniger die exakte GO-ID ausschlaggebend als das konsistente Thema, die Überlappung der Leading-Edge-Gene und die Stabilität über Wiederholungen (etwa Anteil der Simulationen mit FDR < 0.05 und Korrelation der NES-Ränge).

5 Diskussion

Die vorliegende Arbeit untersuchte die Fortpflanzung von Sequenzierfehlern durch eine bioinformatische Standard-Pipeline der RNA-Seq-Analyse. Die durchgeführten Simulationen zeichnen ein ebenso klares wie differenziertes Bild, das für die Interpretation von Transkriptomstudien von erheblicher praktischer Relevanz ist. Dieses Kapitel dient dazu, die zentralen Befunde detailliert zu interpretieren und in den breiteren wissenschaftlichen Kontext einzuordnen. Darüber hinaus werden die Grenzen der gewählten Methodik kritisch reflektiert und darauf aufbauend konkrete Empfehlungen für die zukünftige Forschung sowie für die tägliche Analysepraxis abgeleitet. Der Kern der Erkenntnis, der sich als roter Faden durch die gesamte Diskussion zieht, ist eine fundamentale Dichotomie: Während globale Muster der Genexpression und quantitative Effektstärken eine bemerkenswerte Robustheit aufweisen, erweist sich die binäre, schwellenwertbasierte Entscheidung über die statistische Signifikanz einzelner Gene als überraschend zu stark.

5.1 Detaillierte Interpretation der Ergebnisse

Die zentrale Beobachtung dieser Fallstudie ist, dass die Auswirkungen von Sequenzierfehlern stark von der Betrachtungsebene abhängen. Auf einer Makroebene, die globale Datenstrukturen und aggregierte Effektstärken umfasst, zeigt sich eine hohe Stabilität. Auf der Mikroebene, die sich auf die Signifikanzentscheidung einzelner Gene an einer vordefinierten Schwelle konzentriert, dominiert hingegen die Instabilität. Diese beiden Aspekte sind keine Widersprüche, sondern vielmehr zwei Seiten derselben Medaille, die aus der Natur der Daten, der statistischen Modellierung und der Art der Ergebnisaggregation resultieren.

5.1.1 Die Makroebene: Mechanismen der globalen Robustheit

Die Analysen zeigen eindrücklich, dass die übergeordneten biologischen Signale im Datensatz auch bei künstlich erhöhten Fehlerraten von bis zu 0,1 % pro Base erhalten bleiben. Diese Stabilität ist nicht trivial und lässt sich auf eine Kombination aus der Natur der Messgrößen und den puffernden Eigenschaften der Analysepipeline zurückführen.
An erster Stelle manifestiert sich diese Robustheit in der Konsistenz der Hauptkomponentenanalyse (PCA). Die Proben gruppieren sich in allen Simulationsszenarien klar nach ihrem Spender, was die interindividuelle Variabilität als stärkste Varianzquelle bestätigt. Das durch die fehlerhaften Reads eingebrachte stochastische Rauschen ist demnach nicht stark genug, um die grundlegende Struktur der Daten zu verschleiern. Der Grund hierfür liegt im Wesen der PCA selbst: Sie reduziert die Dimensionalität, indem sie Achsen findet, die die maximale Varianz im Datensatz erklären. Die simulierten Fehler sind per Definition aleatorisch und verteilen sich zufällig über Millionen von Reads und Tausende von Genen. Ihr Einfluss auf die Kovarianzstruktur der Daten ist daher diffus und unsystematisch. Im Gegensatz dazu ist die biologische Varianz zwischen den Spendern ein starkes, systematisches Signal, das Hunderte oder Tausende von Genen kohärent beeinflusst. Die PCA extrahiert dieses dominante Signal mühelos, während das schwache, unkorrelierte Rauschen in den höheren, weniger bedeutsamen Hauptkomponenten untergeht.
Noch aussagekräftiger ist die bemerkenswerte Robustheit der geschätzten $\log_2$-Fold-Changes (LFCs) auf Genebene. Die hohen Korrelationskoeffizienten und die Bland-Altman-Analysen [6], die keinen

systematischen Bias zeigen, belegen dies eindrücklich. Diese Stabilität hat zwei Hauptursachen. Erstens, die Pufferwirkung durch Aggregation. Ein einzelner Sequenzierfehler in einem 100-Basen-Read hat nur einen marginalen Einfluss auf dessen Fähigkeit, korrekt an ein Gen-Feature zu alignieren. Selbst wenn ein Read falsch zugeordnet wird, ist sein Effekt auf die Gesamtzahl der Reads für ein moderat bis stark exprimiertes Gen verschwindend gering. Bei einem Gen mit 1.000 zugeordneten Reads ist ein einzelner fehlgezählter Read eine Störung von nur 0,1 %. Zweitens, die stabilisierenden Eigenschaften des statistischen Modells. Das in DESeq2 [22] verwendete Modell der negativen Binomialverteilung nutzt Informationen über alle Gene hinweg, um die genspezifische Dispersion zu schätzen (Empirical Bayes Shrinkage). Dieser Mechanismus verhindert, dass die Varianzschätzung für einzelne Gene durch zufällige Ausreißer übermäßig beeinflusst wird. Noch wichtiger ist die LFC-Shrinkage, die extreme Fold-Change-Schätzungen, insbesondere bei Genen mit geringer Expression und hoher Varianz, moderiert. Dieser Ansatz wirkt dem Einfluss von zufälligem Rauschen aktiv entgegen, indem er unrealistisch große Effekte, die oft auf zufälligen Zählschwankungen beruhen, zur Null hin korrigiert.

5.1.2 Die Mikroebene: Anatomie der Schwellenfragilität

Im scharfen Kontrast zur globalen Stabilität steht die ausgeprägte Fragilität der Signifikanzentscheidung an der FDR-Schwelle. Die Ergebnisse zeigen, dass die Identität der als differentiell exprimiert klassifizierten Gene (DEGs) empfindlich auf kleinste Störungen in den Eingabedaten reagiert. Dieses Phänomen ist der Kern der Instabilität und verdient eine detaillierte Betrachtung.
Der Mechanismus der sogenannten Flip-Events lässt sich direkt aus der statistischen Testprozedur ableiten. Der p-Wert für ein Gen ist eine Funktion seiner Effektstärke (LFC) und der Unsicherheit dieser Schätzung (Standardfehler des LFC). Wie gezeigt, ist der LFC selbst sehr stabil. Das Problem liegt in der Schätzung seiner Varianz. Schon wenige fehlgezählte Reads, insbesondere bei gering abundanten Genen, können die geschätzte Dispersion leicht erhöhen. Diese erhöhte Varianz führt zu einem größeren Standardfehler des LFC. Im Wald-Test, der den LFC durch seinen Standardfehler teilt, resultiert dies in einer kleineren Teststatistik und somit einem höheren (weniger signifikanten) p-Wert. Für ein Gen, dessen adjustierter p-Wert nahe der Schwelle von 0.05 liegt (z.B. 0.049), genügt diese minimale Erhöhung, um es auf die andere Seite der Schwelle zu befördern (z.B. 0.051).
Die MA-Plots liefern hierfür die visuelle Evidenz. Die charakteristische Trichterform zeigt, dass die LFC-Varianz mit abnehmender mittlerer Genexpression (Abundanz) naturgemäß zunimmt. Gene mit niedrigen Zählwerten sind also von vornherein die Hauptkandidaten für eine hohe statistische Unsicherheit. Sie stellen die Population der Grenzfälle dar, die am anfälligsten für die durch das Rauschen induzierte zusätzliche Varianz-Inflation sind. Die Volcano-Plots bestätigen dies, indem sie eine Ausdünnung der Gendichte direkt oberhalb der FDR-Linie bei steigender Fehlerrate zeigen ein klarer Indikator für einen statistischen Power-Verlust in der kritischen Zone.
Diese Beobachtung hat zudem Konsequenzen für die Prozedur zur Kontrolle der False Discovery Rate. Die p-Wert-Histogramme zeigen eine dosisabhängige Erosion des Signals. Die Häufigkeit sehr kleiner p-Werte nimmt ab, während die der moderaten p-Werte zunimmt. Die Leistungsfähigkeit der Methode nach Benjamini-Hochberg [3] hängt von einem klaren Signal-Anteil in der p-Wert-Verteilung ab. Wenn dieses Signal erodiert, wird die Korrektur konservativer, was den beobachteten Rückgang der DEG-Zahlen und die wechselnde Identität der signifikanten Gene erklärt.

5.1.3 Die Mesoebene: Fortpflanzung der Unsicherheit zur funktionellen Interpretation

Die GSEA [41] stellt eine Brücke zwischen der robusten Makroebene und der fragilen Mikroebene dar und ist daher ein besonders aufschlussreicher Endpunkt der Analyse. Da sie auf der Rangfolge aller Gene basiert, profitiert sie von der hohen Stabilität der LFC-Werte, die diese Rangfolge primär bestim-

men. Dies erklärt, warum breit getragene, übergeordnete biologische Themen in den Ergebnissen stabil bleiben. Eine Funktion wie Metabolismus, die von Hunderten von Genen mit kohärenten, wenn auch kleinen Änderungen getragen wird, ist unempfindlich gegenüber der Verschiebung einzelner Gene in der Rangliste.
Die Instabilität manifestiert sich jedoch bei der Betrachtung einzelner GO-Terme [1]. Die Signifikanz eines Gen-Sets in der GSEA wird maßgeblich durch die Gene an den Extremen der Rangliste bestimmt die sogenannte Leading Edge. Die zuvor beschriebenen Flip-Events und die allgemeine Rang-Instabilität von Genen nahe der statistischen Nachweisgrenze verändern die Zusammensetzung genau dieser Leading Edge. Wenn die Anreicherung eines GO-Terms von einer kleinen Gruppe von Genen abhängt, die selbst nur grenzwertig signifikant oder gering abundant sind, wird die Signifikanz dieses Terms (ausgedrückt durch den NES und den FDR-Wert) ebenfalls instabil. Dies erklärt präzise die beobachtete Rang- und Signifikanzdrift einzelner GO-Terme. Aus diesen Beobachtungen lässt sich eine interpretatorische Hierarchie der Robustheit ableiten:

1. Höchste Robustheit: Globale Muster wie die Proben-Clusterung in der PCA.
2. Hohe Robustheit: Quantitative Effektstärken (LFCs) und breit getragene funktionelle Themen (GSEA-Cluster).
3. Geringe Robustheit: Die exakte Zusammensetzung und Größe einer DEG-Liste sowie die exakte Rangposition und Signifikanz einzelner Gene oder spezifischer GO-Terme.

5.2 Einordnung in den wissenschaftlichen Kontext

Die Erkenntnis, dass bioinformatische Analyseentscheidungen einen erheblichen Einfluss auf die finalen Ergebnisse haben, ist in der Forschungsgemeinschaft fest verankert. Großangelegte Konsortialprojekte wie SEQC/MAQC-III haben die Bedeutung von Plattform-, Protokoll- und Pipeline-Wahlen für die Reproduzierbarkeit eindrücklich belegt [36]. Systematische Vergleiche von Alignern oder Quantifizierungstools kamen zu ähnlichen Schlussfolgerungen [11, 2].
Die vorliegende Arbeit knüpft an diese Tradition an, wählt jedoch einen spezifischeren und fundamentaleren Ansatzpunkt. Anstatt verschiedene bioinformatische Werkzeuge (Pipeline A vs. Pipeline B) zu vergleichen, isoliert diese Studie explizit die Qualität der initialen Rohdaten als eine der ursprünglichsten Fehlerquellen und verfolgt deren Dosis-Wirkungs-Beziehung innerhalb einer fixierten Pipeline. Sie liefert damit eine mechanistische Perspektive auf die Fragilität von Ergebnissen, die über einen reinen Werkzeugvergleich hinausgeht.
Der zentrale Befund dieser Arbeit, die höhere Robustheit von Effektstärken gegenüber p-Wert-basierten Signifikanzentscheidungen ist ein direkter Beitrag zur Debatte um die Replikationskrise. Die Studie liefert damit ein empirisches Argument aus der Transkriptomik, dass die Forderung nach einer stärkeren Betonung quantitativer Effekte und deren Unsicherheitsmaße untermauert. Sie zeigt, wie eine alleinige Fixierung auf die binäre Signifikanz zu scheinbar widersprüchlichen Ergebnissen führen kann, selbst wenn die zugrundeliegenden quantitativen Effekte nahezu identisch sind.
Darüber hinaus stellen die Ergebnisse eine wichtige Ergänzung zu Studien dar, die die Notwendigkeit einer ausreichenden Anzahl biologischer Replikate betonen [35]. Während diese Studien argumentieren, dass mehr Replikate zur besseren Schätzung der biologischen Varianz und somit zur Erhöhung der statistischen Power notwendig sind, zeigt unsere Arbeit einen komplementären Aspekt: Mehr Replikate erhöhen auch die Robustheit gegenüber technischem Rauschen. Durch die Verringerung des Standardfehlers der Schätzungen entfernen sie mehr Gene von der instabilen Kipp-Schwelle des p-Wertes, wodurch die gesamte Analyse widerstandsfähiger gegen die hier untersuchten Störungen wird.

5.3 Kritische Reflexion der Methodik und Limitationen

Trotz klarer Ergebnisse unterliegt die Studie Limitationen, die ihre Generalisierbarkeit einschränken. Die bewusste Entscheidung für ein kontrolliertes, aber vereinfachtes Design war notwendig, um die Fehlerfortpflanzung isoliert untersuchen zu können, impliziert aber auch spezifische Einschränkungen.

5.3.1 Der Fallstudiencharakter

Die gesamte Untersuchung basiert auf einem einzigen, wenn auch gut charakterisierten Datensatz [14, 12]. Dessen günstiges Design (gepaarte Spender, solide Sequenziertiefe) reduziert die biologische Hintergrundvarianz erheblich. In einem Experiment mit höherer biologischer Varianz (z.B. ungepaarte Proben aus einer heterogenen Population) oder geringerer Sequenziertiefe wäre das Signal-Rausch-Verhältnis von vornherein schlechter. Es ist plausibel anzunehmen, dass die beobachtete Schwellenfragilität in einem solchen Szenario noch stärker ausgeprägt wäre, da mehr Gene im Bereich geringer Abundanz und statistischer Unsicherheit liegen würden.

5.3.2 Das idealisierte Fehlermodell

Eine wesentliche Limitation liegt im bewusst minimalistischen Fehlermodell. Es wurden ausschließlich zufällige Basensubstitutionen simuliert. Reale Sequenzierfehler umfassen jedoch auch Insertions- und Deletionsfehler (Indels), die für Aligner wie STAR [10] weitaus problematischer sind. Ein Indel kann zu einem teilweisen (Soft-Clipping) oder vollständigen Verlust des Alignments eines Reads führen, was einen ungleich größeren Effekt auf die Zählmatrix hat als eine einzelne Falschinformation. Des Weiteren sind reale Illumina-Fehlerraten nicht uniform, sondern zeigen systematische, positions- und kontextabhängige Muster [31]. Die hier präsentierten Ergebnisse sollten daher als eine konservative untere Schranke für die in der Realität zu erwartende Instabilität verstanden werden.

5.3.3 Die Black-Box-Natur der Pipeline

Die Studie behandelt die Analysepipeline als eine Black Box und misst die Veränderung zwischen Input und Output. Dies ist ein valider und leistungsfähiger Ansatz der Monte-Carlo-Simulation. Er erklärt jedoch nicht die detaillierten Interaktionen zwischen den Fehlern und den einzelnen Algorithmen. Eine weiterführende Analyse könnte untersuchen, wie genau STAR [10] oder featureCounts [21] auf bestimmte Fehlertypen reagieren, um die Fehlerfortpflanzung auf einer noch granulareren Ebene zu verstehen.

6 Fazit und Ausblick

6.1 Fazit

Die Arbeit hat zeigt, wie sich Sequenzierfehler durch eine RNA-Seq-Pipeline auswirken. Zwei Punkte sind entscheidend. Erstens bleiben globale Muster wie Proben-Clusterung in der Hauptkomponentenanalyse (PCA) und $\log_2$-Fold-Changes gegenüber zufälligen Fehlern weitgehend stabil und Konsistenz. Zweitens, wenn wir ganz nah andere messbare Werten (z.B.: p-Wert oder MA-Plot) betrachten, ist es sichbar, dass wir eine statistische Instabilität merken können. Diese Instabilität spiegelt eine ausgeprägte Empfindlichkeit wider und könnte bis auf die funktionelle Ebene reichen. In der Praxis könnte heißen, dass das Effektstärken bei der Priorisierung stärker als starre p-Wert-Grenzen gewichtet werden sollten. Zusätzlich sollten Stabilitätsangaben mitberichtet werden, etwa Überlappung der DEG-Listen, Flip-Rate oder Übereinstimmungsmaße. Die Qualität der Rohdaten beeinflusst damit nicht nur die Zahl der Treffer, sondern auch deren Identität.

6.2 Ausblick

Aus den Erkenntnissen und Limitationen dieser Arbeit ergeben sich konkrete Schlussfolgerungen. Der übergeordnete Gedanke ist, Stabilität nicht als ein zufälliges Nebenprodukt, sondern als ein explizites und messbares Ziel von Transkriptomanalysen zu etablieren.
Zukünftige Simulationsstudien sollten auf den hier gelegten Grundlagen aufbauen und die Limitationen gezielt adressieren. Dies umfasst die Verwendung von realistischeren Fehlermodellen, die Indels und systematische Bias-Profile beinhalten. Besonders vielversprechend ist der Vergleich der Robustheit unterschiedlicher Pipeline-Architekturen. So könnte man hypothesieren, dass alignmentfreie Methoden wie Salmon [29] oder Kallisto [7], die auf k-mer-Zählungen basieren, empfindlicher auf Basensubstitutionen reagieren (da diese k-mere zerstören), aber möglicherweise robuster gegenüber Fehlern sind, die die exakte Alignment-Position beeinflussen. Solche vergleichenden Stabilitätsanalysen sind ein wichtiger nächster Schritt.
Die Botschaft für die Praxis ist klar. Effektstärken und übergeordnete biologische Themen sind weitgehend robuste Ergebnisse. Grenzfälle an einer hart definierten Signifikanzschwelle sind es nicht. Die Interpretation sollte sich primär auf die Richtung und Größe der LFCs stützen. MA- und Volcano-Plots sind keine optionalen, sondern essenzielle Werkzeuge, um ein Gefühl für die Verteilung und die mit der Abundanz verbundene Unsicherheit zu bekommen. Eine Liste signifikanter Gene sollte niemals isoliert präsentiert werden. Sie muss durch globale QC-Metriken und wo immer möglich, durch eine explizite Stabilitätsbewertung kontextualisiert werden. Die routinemäßige Angabe von Stabilitätsmaßen wie dem Jaccard-Index zwischen Bootstrap- oder Subsampling-Wiederholungen sollte zum Standard werden. Bei der Ergebnispräsentation sollte dem Prinzip der abnehmenden Robustheit gefolgt werden. Beginnen Sie mit den stabilsten Befunden (globale Muster, robuste Effektstärken) und arbeiten Sie sich zu den fragileren Details (exakte DEG-Listen, einzelne GO-Ränge) vor. Wer wissenschaftliche oder klinische Entscheidungen auf Basis von Transkriptomdaten trifft, muss sich der inhärenten Fragilität von schwellenwertbasierten Ergebnissen bewusst sein. Die aktive Messung und Berichterstattung von Stabilität ist kein Mehraufwand, sondern ein notwendiger Schritt hin zu einer transparenteren und reproduzierbareren Wissenschaft.

Anhang

Tabelle 1: Proben-Metadaten und SRA-Statistiken (GSE52778).

Run	GSM (GEO)	BioSample	SRX (Experiment)	Spender (cell_line)	Behandlung	AvgSpotLen	Bases [G]	Bytes [Gb]
SRR1039508	GSM1275862	SAMN02422669	SRX384345	N61311	Untreated	126	2.89	1.55
SRR1039509	GSM1275863	SAMN02422675	SRX384346	N61311	Dexamethasone	126	2.67	1.45
SRR1039510	GSM1275864	SAMN02422668	SRX384347	N61311	Albuterol	126	2.88	1.56
SRR1039511	GSM1275865	SAMN02422667	SRX384348	N61311	Albuterol_Dexamethasone	126	2.76	1.48
SRR1039512	GSM1275866	SAMN02422678	SRX384349	N052611	Untreated	126	3.55	2.01
SRR1039513	GSM1275867	SAMN02422670	SRX384350	N052611	Dexamethasone	87	3.79	2.22
SRR1039514	GSM1275868	SAMN02422681	SRX384351	N052611	Albuterol	126	5.04	3.03
SRR1039515	GSM1275869	SAMN02422671	SRX384352	N052611	Albuterol_Dexamethasone	114	3.38	1.86
SRR1039516	GSM1275870	SAMN02422682	SRX384353	N080611	Untreated	120	3.61	2.10
SRR1039517	GSM1275871	SAMN02422673	SRX384354	N080611	Dexamethasone	126	4.32	2.59
SRR1039518	GSM1275872	SAMN02422679	SRX384355	N080611	Albuterol	126	3.84	2.30
SRR1039519	GSM1275873	SAMN02422672	SRX384356	N080611	Albuterol_Dexamethasone	107	3.38	1.96
SRR1039520	GSM1275874	SAMN02422683	SRX384357	N061011	Untreated	101	3.52	2.06
SRR1039521	GSM1275875	SAMN02422677	SRX384358	N061011	Dexamethasone	98	4.07	2.39
SRR1039522	GSM1275876	SAMN02422680	SRX384359	N061011	Albuterol	125	3.56	1.96
SRR1039523	GSM1275877	SAMN02422674	SRX384360	N061011	Albuterol_Dexamethasone	126	3.92	2.22

1

[1] Die Spalten (Run, GSM (GEO), BioSample, SRX (Experiment), Spender (cell_line), Behandlung, AvgSpotLen, Bases [G], Bytes [Gb]) sind in den zugehörigen SRA-/GEO-Metadaten erläutert; die vollständige Run-/Metadaten-Tabelle ist online im SRA/Traces-View des BioProject PRJNA229998 sowie im GEO-Eintrag GSE52778 verfügbar. [32, 12]

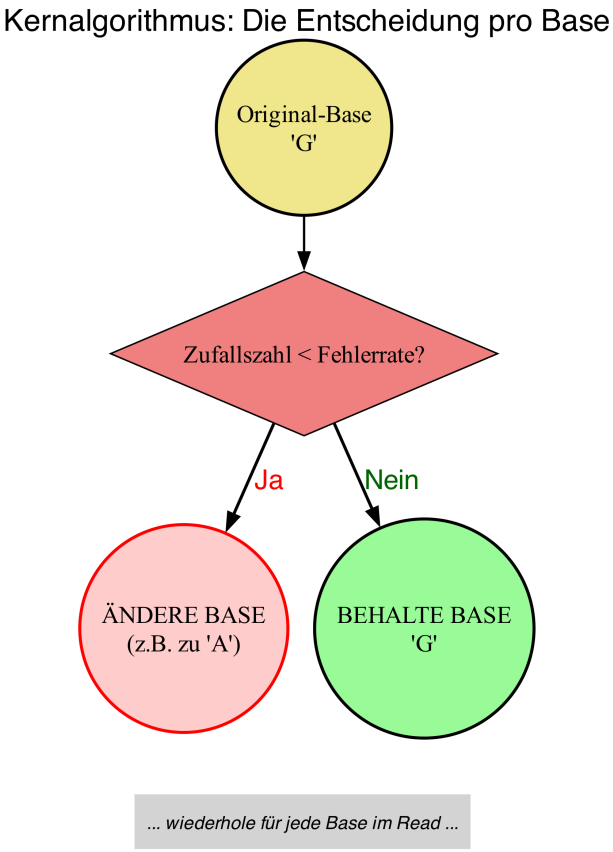

Abbildung 1: Vereinfachte Darstellung des Fehlersimulations-Algorithmus. Jede Base der Originalsequenz wird mit einer Wahrscheinlichkeit ϵ durch eine andere Base ersetzt.

Listing 1: introduce_fastq_errors.py

```
# ========================================
# FileName: introduce_fastq_errors.py
# Date: 22:35 08.Mai.2025
# Author: Marcos Chow Castro
# Brief: Fügt simulierte Sequenzierungsfehler in Paired-End FASTQ-Dateien ein,
#        loggt den Prozess und speichert die Ausgabe als komprimierte .gz-Dateien.
# ========================================

'''
Beispielaufruf:
python scripts/introduce_fastq_errors.py \
    --r1_in data/raw/SRR1039508_1.fastq.gz \
    --r2_in data/raw/SRR1039508_2.fastq.gz \
    --output_dir data/raw_error \
    --r1_out_name SRR1039508_1_err.fastq.gz \
    --r2_out_name SRR1039508_2_err.fastq.gz \
    --error_rate 0.001 \
    --log_file introduce_fastq_errors.log
'''

import random
import argparse
import gzip
import os
```

```
25 import sys
26 import logging
27
28 # Globaler Logger wird hier initialisiert, damit er in Funktionen verfügbar ist
29 logger = logging.getLogger(__name__)
30
31 def setup_logging(log_file_path):
32     """Konfiguriert das Logging-System."""
33     logger.setLevel(logging.INFO) # Minimales Level für den Logger selbst
34
35     # Formatter für die Log-Nachrichten
36     formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s', datefmt='%Y↩
    -%m-%d %H:%M:%S')
37
38     # File Handler - schreibt Logs in eine Datei (Append-Modus)
39     try:
40         file_handler = logging.FileHandler(log_file_path, mode='a', encoding='utf-8')
41         file_handler.setLevel(logging.INFO) # Level für diesen Handler
42         file_handler.setFormatter(formatter)
43         logger.addHandler(file_handler)
44     except Exception as e:
45         print(f"Kritischer Fehler beim Einrichten des File-Loggings nach {log_file_path}: ↩
    {e}", file=sys.stderr)
46
47
48     console_handler = logging.StreamHandler(sys.stdout)
49     console_handler.setLevel(logging.INFO)
50     console_handler.setFormatter(formatter)
51     logger.addHandler(console_handler)
52
53 def introduce_errors_to_fastq(sequence, error_rate=0.001):
54     """Fügt Fehler in eine einzelne Sequenz ein."""
55     bases = ['A', 'C', 'G', 'T']
56     new_sequence = list(sequence)
57     errors_introduced_in_read = 0
58     for i in range(len(new_sequence)):
59         if random.random() < error_rate:
60             original_base = new_sequence[i].upper()
61             possible_new_bases = [b for b in bases if b != original_base]
62             if not possible_new_bases or original_base not in bases: # Auch 'N' etc. ↩
    abfangen
63                 new_base = random.choice(bases)
64             else:
65                 new_base = random.choice(possible_new_bases)
66
67             new_sequence[i] = new_base
68             errors_introduced_in_read += 1
69     return "".join(new_sequence), errors_introduced_in_read
70
71 def process_fastq_pair(input_r1_path, input_r2_path,
72                        output_dir, output_r1_name, output_r2_name,
73                        error_rate=0.001):
74     """
75     Verarbeitet ein Paar von FASTQ-Dateien, führt Fehler ein und speichert sie
76     komprimiert im angegebenen Output-Verzeichnis.
```

```
77      """
78
79      # 1. Output-Verzeichnis erstellen, falls es nicht existiert
80      try:
81          os.makedirs(output_dir, exist_ok=True)
82          logger.info(f"Output-Verzeichnis: '{output_dir}'")
83      except OSError as e:
84          logger.error(f"Konnte Output-Verzeichnis '{output_dir}' nicht erstellen: {e}")
85          sys.exit(1)
86
87      # 2. Vollständige Pfade für Output-Dateien erstellen
88      full_output_r1_path = os.path.join(output_dir, output_r1_name)
89      full_output_r2_path = os.path.join(output_dir, output_r2_name)
90
91      # Sicherstellen, dass Output-Namen auf .gz enden
92      if not full_output_r1_path.endswith(".gz"):
93          logger.warning(f"Output-Datei R1 '{full_output_r1_path}' endet nicht mit .gz. Sie ↩
        wird trotzdem als .gz komprimiert.")
94      if not full_output_r2_path.endswith(".gz"):
95          logger.warning(f"Output-Datei R2 '{full_output_r2_path}' endet nicht mit .gz. Sie ↩
        wird trotzdem als .gz komprimiert.")
96
97
98      total_bases_r1 = 0
99      total_errors_r1 = 0
100     total_bases_r2 = 0
101     total_errors_r2 = 0
102     reads_processed = 0
103
104     logger.info(f"Lese R1 von: {input_r1_path}")
105     logger.info(f"Lese R2 von: {input_r2_path}")
106     logger.info(f"Schreibe komprimiertes R1 nach: {full_output_r1_path}")
107     logger.info(f"Schreibe komprimiertes R2 nach: {full_output_r2_path}")
108     logger.info(f"Angestrebte Fehlerrate: {error_rate}")
109
110     try:
111         # Gzip-kompatibles Öffnen für Input, Gzip-Öffnen für komprimierten Output
112         with (gzip.open(input_r1_path, 'rt', encoding='utf-8') if input_r1_path.endswith("↩
        .gz") else open(input_r1_path, 'r', encoding='utf-8')) as infile_r1, \
113                 (gzip.open(input_r2_path, 'rt', encoding='utf-8') if input_r2_path.endswith("↩
        .gz") else open(input_r2_path, 'r', encoding='utf-8')) as infile_r2, \
114              gzip.open(full_output_r1_path, 'wt', encoding='utf-8') as outfile_r1, \
115              gzip.open(full_output_r2_path, 'wt', encoding='utf-8') as outfile_r2:
116
117             while True:
118                 header_r1 = infile_r1.readline()
119                 if not header_r1:
120                     break
121                 seq_r1 = infile_r1.readline().strip()
122                 plus_r1 = infile_r1.readline()
123                 qual_r1 = infile_r1.readline().strip()
124
125                 header_r2 = infile_r2.readline()
126                 if not header_r2:
127                     logger.warning("R2-Datei unerwartet zu Ende. Stelle sicher, dass R1 ↩
```

```
    und R2 die gleiche Anzahl Reads haben.")
128                     break
129                 seq_r2 = infile_r2.readline().strip()
130                 plus_r2 = infile_r2.readline()
131                 qual_r2 = infile_r2.readline().strip()
132
133                 modified_seq_r1, errors_in_r1 = introduce_errors_to_fastq(seq_r1, ↩
    error_rate)
134                 modified_seq_r2, errors_in_r2 = introduce_errors_to_fastq(seq_r2, ↩
    error_rate)
135
136                 total_bases_r1 += len(seq_r1)
137                 total_errors_r1 += errors_in_r1
138                 total_bases_r2 += len(seq_r2)
139                 total_errors_r2 += errors_in_r2
140                 reads_processed +=1
141
142                 outfile_r1.write(header_r1)
143                 outfile_r1.write(modified_seq_r1 + '\n')
144                 outfile_r1.write(plus_r1)
145                 outfile_r1.write(qual_r1 + '\n')
146
147                 outfile_r2.write(header_r2)
148                 outfile_r2.write(modified_seq_r2 + '\n')
149                 outfile_r2.write(plus_r2)
150                 outfile_r2.write(qual_r2 + '\n')
151
152                 if reads_processed > 0 and reads_processed % 100000 == 0:
153                     logger.info(f"Verarbeitet {reads_processed} Read-Paare...")
154
155         # Endstatistiken
156         logger.info(f"\nVerarbeitung abgeschlossen für {input_r1_path} und {input_r2_path↩
    }.")
157         logger.info("--- Read 1 Statistik ---")
158         logger.info(f"Gesamtzahl Basen R1: {total_bases_r1}")
159         logger.info(f"Eingefügte Fehler R1: {total_errors_r1}")
160         if total_bases_r1 > 0:
161             logger.info(f"Effektive Fehlerrate R1: {total_errors_r1 / total_bases_r1:.6f}"↩
    )
162
163         logger.info("\n--- Read 2 Statistik ---")
164         logger.info(f"Gesamtzahl Basen R2: {total_bases_r2}")
165         logger.info(f"Eingefügte Fehler R2: {total_errors_r2}")
166         if total_bases_r2 > 0:
167             logger.info(f"Effektive Fehlerrate R2: {total_errors_r2 / total_bases_r2:.6f}"↩
    )
168
169     except FileNotFoundError as e:
170         logger.error(f"Input-Datei nicht gefunden: {e.filename}")
171         sys.exit(1)
172     except Exception as e:
173         logger.error(f"Ein unerwarteter Fehler ist aufgetreten: {e}", exc_info=True)
174         sys.exit(1)
175
176
```

```
177 if __name__ == "__main__":
178     parser = argparse.ArgumentParser(
179         description="Führt Sequenzierungsfehler in Paired-End FASTQ-Dateien ein und speichert das Ergebnis komprimiert. Liest .fastq und .fastq.gz Dateien, schreibt komprimierte .fastq.gz Dateien.",
180         formatter_class=argparse.RawTextHelpFormatter
181     )
182     parser.add_argument("--r1_in", required=True, help="Eingabe FASTQ-Datei R1 (kann .gz sein)")
183     parser.add_argument("--r2_in", required=True, help="Eingabe FASTQ-Datei R2 (kann .gz sein)")
184 
185     parser.add_argument("--output_dir", required=True,
186                         help="Verzeichnis, in dem die modifizierten, komprimierten FASTQ-Dateien gespeichert werden.\nWird erstellt, falls es nicht existiert.")
187 
188     parser.add_argument("--r1_out_name", required=True,
189                         help="Dateiname für die modifizierte R1 FASTQ-Datei (z.B. 'sample_1_err.fastq.gz').\nWird komprimiert im --output_dir gespeichert.")
190     parser.add_argument("--r2_out_name", required=True,
191                         help="Dateiname für die modifizierte R2 FASTQ-Datei (z.B. 'sample_2_err.fastq.gz').\nWird komprimiert im --output_dir gespeichert.")
192 
193     parser.add_argument("--error_rate", type=float, default=0.001,
194                         help="Wahrscheinlichkeit für einen Fehler pro Base (Default: 0.001, d.h. 1 in 1000).")
195 
196     parser.add_argument("--log_file", default="introduce_fastq_errors.log",
197                         help="Pfad zur Log-Datei (Default: introduce_fastq_errors.log im aktuellen Verzeichnis).")
198 
199     # Wenn keine Argumente gegeben werden (oder -h/--help), zeige Hilfe und beende.
200     if len(sys.argv) == 1:
201         parser.print_help(sys.stderr)
202         sys.exit(1)
203 
204     args = parser.parse_args()
205 
206     setup_logging(args.log_file)
207 
208     logger.info("Skript gestartet mit folgenden Argumenten:")
209     for arg, value in vars(args).items():
210         logger.info(f"  --{arg}: {value}")
211 
212     try:
213         process_fastq_pair(
214             args.r1_in, args.r2_in,
215             args.output_dir, args.r1_out_name, args.r2_out_name,
216             args.error_rate
217         )
218         logger.info("Skript erfolgreich beendet.")
219     except SystemExit: # Fängt sys.exit() ab, damit das Log noch geschrieben wird
220         logger.error("Skript vorzeitig beendet aufgrund eines Fehlers (siehe vorherige Log-Meldungen).")
221         raise # Erneutes Auslösen, um den Exit-Code beizubehalten
```

```
222     except Exception as e:
223         logger.critical(f"Ein unerwarteter Fehler auf oberster Ebene ist aufgetreten: {e}", exc_info=True)
224         sys.exit(1) # Sicherstellen, dass bei unerwarteten Fehlern ein Fehlercode zurückgegeben wird
225
226 # vim: set fdm=marker:
```

Listing 2: fehlerfortpflanzung_vergleich.R

```
1  ################################################################################
2  # Dateiname: fehlerfortpflanzung_vergleich.R
3  # Zweck:     Führt eine vollständige DESeq2-Analyse und Robustheitsprüfung durch,
4  #            erstellt alle diagnostischen und vergleichenden Plots mit
5  #            Formeln/Erklärungen und fasst sie in einem finalen PDF zusammen.
6  # Autor:     Marcos Chow Castro
7  # Datum:     23. Juli 2025
8  ################################################################################
9
10 #  0. Initialisierung & Setup
11 if (!require("pacman")) install.packages("pacman", repos = "https://cloud.r-project.org/")
12 pacman::p_load(
13     "DESeq2", "BiocParallel", "dplyr", "tibble", "ggplot2", "pheatmap", "scales",
14     "AnnotationDbi", "org.Hs.eg.db", "ggvenn", "ggalt", "patchwork", "clusterProfiler",
15     "enrichplot", "rstudioapi", "umap", "grid"
16 )
17
18 base_project_dir <- getwd()
19 message("PROJEKT-BASISVERZEICHNIS: ", base_project_dir)
20
21 #  Pfade und Verzeichnisse
22 standard_counts_path <- file.path(base_project_dir, "counts", "GSE52778_raw_counts_trimmed.tsv")
23 manipulated_counts_path <- file.path(base_project_dir, "counts", "counts_error", "GSE52778_raw_counts_trimmed_error0001.tsv")
24 main_output_dir <- file.path(base_project_dir, "output")
25 standard_output_dir <- file.path(main_output_dir, "standard")
26 manipulated_output_dir <- file.path(main_output_dir, "manipulated")
27 final_output_dir <- file.path(base_project_dir, "output_final")
28 dir.create(standard_output_dir, recursive = TRUE, showWarnings = FALSE)
29 dir.create(manipulated_output_dir, recursive = TRUE, showWarnings = FALSE)
30 dir.create(final_output_dir, recursive = TRUE, showWarnings = FALSE)
31
32 #  Parallele Verarbeitung
33 available_cores <- parallel::detectCores()
34 N_CORES <- if (!is.na(available_cores) && available_cores > 1) min(8, available_cores - 1) else 1
35 BP_PARAM <- if (N_CORES > 1 && .Platform$OS.type == "unix") MulticoreParam(workers = N_CORES) else SerialParam()
36 register(BP_PARAM)
37 message("[INFO] Verwende ", bpnworkers(bpparam()), " Kerne mit ", class(bpparam())[1], " Backend.")
38
39 ################################################################################
40 #  TEIL 1: ANALYSEFUNKTION
```

```
41 ################################################################################
42 run_deseq_analysis <- function(count_file_path, output_dir, analysis_name) {
43     message(paste(" Starte DESeq2 Analyse für:", analysis_name, ""))
44     plots_output_dir <- file.path(output_dir, "plots")
45     dir.create(plots_output_dir, showWarnings = FALSE, recursive = TRUE)
46 
47     #  Daten einlesen & DESeq2 Analyse
48     if (!file.exists(count_file_path)) stop("FEHLER: Zählungsdatei nicht gefunden: ", ↩
       count_file_path)
49     count_df <- read.table(count_file_path, header = TRUE, sep = "\t", comment.char = "#",↩
        check.names = FALSE)
50     srr2gsm_full <- c("SRR1039508"="GSM1275862","SRR1039509"="GSM1275863","SRR1039510"="↩
       GSM1275864","SRR1039511"="GSM1275865","SRR1039512"="GSM1275866","SRR1039513"="↩
       GSM1275867","SRR1039514"="GSM1275868","SRR1039515"="GSM1275869","SRR1039516"="↩
       GSM1275870","SRR1039517"="GSM1275871","SRR1039518"="GSM1275872","SRR1039519"="↩
       GSM1275873","SRR1039520"="GSM1275874","SRR1039521"="GSM1275875","SRR1039522"="↩
       GSM1275876","SRR1039523"="GSM1275877")
51     gsm_meta_full <- data.frame(row.names = names(srr2gsm_full), donor = factor(rep(c("↩
       N61311", "N052611", "N080611", "N061011"), each = 4)), condition = factor(rep(c("↩
       untreated", "Dex", "Alb", "Alb_Dex"), times = 4), levels = c("untreated", "Dex", "Alb"↩
       , "Alb_Dex")))
52     expected_sample_names <- rownames(gsm_meta_full)
53     meta_cols_fc <- c("Geneid", "Chr", "Start", "End", "Strand", "Length")
54     sample_cols_in_df <- colnames(count_df)[!(colnames(count_df) %in% meta_cols_fc)]
55     mat_raw <- as.matrix(count_df[, sample_cols_in_df, drop = FALSE])
56     rownames(mat_raw) <- count_df$Geneid
57     if (ncol(mat_raw) != length(expected_sample_names)) { stop(paste("FEHLER: Proben-↩
       Spalten (", ncol(mat_raw), ") in", basename(count_file_path), "stimmt nicht mit ↩
       erwarteter Anzahl (", length(expected_sample_names), ") überein.")) }
58     colnames(mat_raw) <- expected_sample_names
59     mode(mat_raw) <- "integer"; mat_raw[is.na(mat_raw) | mat_raw < 0] <- 0L
60     mat_raw <- mat_raw[!is.na(rownames(mat_raw)) & rownames(mat_raw) != "" & !duplicated(↩
       rownames(mat_raw)), ]
61     col_data <- gsm_meta_full
62     if (!identical(rownames(col_data), colnames(mat_raw))) stop("INTERNER FEHLER: ↩
       Konsistenzprüfung mat_raw/col_data fehlgeschlagen.")
63     dds <- DESeqDataSetFromMatrix(countData = mat_raw, colData = col_data, design = ~ ↩
       donor + condition)
64     dds <- dds[rowSums(counts(dds)) >= 10, ]
65     dds <- DESeq(dds, BPPARAM = BP_PARAM)
66     res <- results(dds, contrast = c("condition", "Dex", "untreated"), BPPARAM = BP_PARAM)
67     res_df <- as.data.frame(res) %>% rownames_to_column("Symbol") %>% mutate(padj = ifelse↩
       (is.na(padj), 1, padj)) %>% arrange(padj, pvalue)
68     ens_ids <- mapIds(org.Hs.eg.db, keys = res_df$Symbol, column = "ENSEMBL", keytype = "↩
       SYMBOL", multiVals = "first")
69     res_df$Ensembl <- ens_ids
70     res_df <- res_df %>% select(Symbol, Ensembl, everything())
71     results_filename <- file.path(output_dir, paste0("DESeq2_results_", analysis_name, ".↩
       txt"))
72     write.table(res_df, results_filename, sep = "\t", row.names = FALSE, quote = FALSE)
73     message("Ergebnis-TXT-Datei gespeichert: ", results_filename)
74 
75     #  Plots erstellen und Objekte speichern
76     # MA-Plot mit ggplot2
77     ma_data <- as.data.frame(res) %>%
```

```
78          mutate(significant = ifelse(is.na(padj) | padj >= 0.1, "Nein", "Ja"))
79
80      ma_plot <- ggplot(ma_data, aes(x = baseMean, y = log2FoldChange)) +
81          geom_point(aes(color = significant), alpha = 0.5) +
82          scale_x_log10() + scale_color_manual(name = "Significant (padj < 0.1)", values = c("Ja" = "blue", "Nein" = "grey")) +
83          geom_hline(yintercept = 0, linetype = "dashed") + theme_minimal() +
84          labs(title = paste("MA-Plot:", analysis_name),
85               caption = bquote(atop(bold("Formel:") ~ M == log[2](A)-log[2](B) ~";"~ A == frac(1,2)(log[2](A)+log[2](B)),
86                                     bold("Kommentar:") ~ "Zeigt Expressionsänderung (M) vs. mittlere Expression (A). Hilft, intensitätsabhängige Effekte zu erkennen.")))
87      ggsave(file.path(plots_output_dir, paste0(analysis_name, "_MA_plot.png")), ma_plot, width = 8, height = 7)
88      # extra png
89      ggsave(file.path(final_output_dir, paste0(analysis_name, "_MA_plot.png")), ma_plot, width = 8, height = 7)
90
91      volcano_plot_data <- res_df %>% mutate(significant = ifelse(padj < 0.05 & abs(log2FoldChange) > 1, "Ja", "Nein"))
92      volcano_plot <- ggplot(volcano_plot_data, aes(x = log2FoldChange, y = -log10(pvalue))) +
93        geom_point(aes(color = significant), alpha = 0.5) + scale_color_manual(values = c("Ja" = "red", "Nein" = "grey")) +
94        theme_minimal() + labs(title = paste("Volcano Plot:", analysis_name), caption = bquote(atop(bold("Formel:") ~ y == -log[10]("p-Wert"), bold("Kommentar:") ~ "Kombiniert statistische Signifikanz (y) mit biologischer Veränderung (x, log2FC)."))
95      ggsave(file.path(plots_output_dir, paste0(analysis_name, "_Volcano_plot.png")), volcano_plot, width = 8, height = 7)
96      # extra png
97      ggsave(file.path(final_output_dir, paste0(analysis_name, "_Volcano_plot.png")), volcano_plot, width = 8, height = 7)
98
99      vsd <- vst(dds, blind = FALSE)
100
101     pca_plot <- plotPCA(vsd, intgroup = c("condition", "donor")) +
102         labs(title = paste("PCA Plot:", analysis_name), caption = bquote(atop(bold("Formel:") ~ C == X^T*X ~ "(Kovarianzmatrix)", bold("Kommentar:") ~ "Reduziert Dimensionen durch Finden der Hauptvarianzachsen (PCs), um Proben-Cluster zu visualisieren."))
103     ggsave(file.path(plots_output_dir, paste0(analysis_name, "_PCA_plot.png")), pca_plot, width = 8, height = 7)
104     # extra png
105     ggsave(file.path(final_output_dir, paste0(analysis_name, "_PCA_plot.png")), pca_plot, width = 8, height = 7)
106
107     message("Erstelle UMAP-Plot...")
108     umap_results <- umap::umap(t(assay(vsd)))
109     umap_df <- as.data.frame(umap_results$layout) %>% setNames(c("UMAP1", "UMAP2")) %>% cbind(colData(vsd))
110     umap_plot <- ggplot(umap_df, aes(x = UMAP1, y = UMAP2, color = condition, shape = donor)) +
111         geom_point(size = 3) + theme_minimal() +
112         labs(title = paste("UMAP Plot:", analysis_name), caption = "Kommentar: Nichtlineare Dimensionsreduktion, die sowohl lokale als auch globale Strukturen bewahrt, um Proben-Cluster zu zeigen.")
```

```
113     ggsave(file.path(plots_output_dir, paste0(analysis_name, "_UMAP_plot.png")), umap_plot, width = 8, height = 7)
114     # extra png
115     ggsave(file.path(final_output_dir, paste0(analysis_name, "_UMAP_plot.png")), umap_plot, width = 8, height = 7)
116
117     message("Erstelle Heatmap...")
118     top_gene_symbols <- res_df %>% arrange(padj, pvalue) %>% head(30) %>% pull(Symbol)
119     heatmap_matrix <- assay(vsd)[top_gene_symbols, ]; heatmap_matrix_scaled <- t(scale(t(heatmap_matrix))); heatmap_matrix_scaled[is.na(heatmap_matrix_scaled)] <- 0
120     heatmap_filename <- file.path(plots_output_dir, paste0(analysis_name, "_Heatmap_Top30_DEGs.png"))
121     pheatmap_obj <- pheatmap::pheatmap(heatmap_matrix_scaled, annotation_col = as.data.frame(colData(vsd)[, c("condition", "donor")]),
122         main = paste("Heatmap: Top 30 DEGs für", analysis_name,
123                      "\nFormel: Z = (x - mu) / sigma (Z-Score pro Gen)",
124                      "\nKommentar: Visualisiert relative Expressionslevel zur Mustererkennung."),
125         filename = heatmap_filename, width = 10, height = 8)
126     # extra png
127     ggsave(file.path(final_output_dir, paste0(analysis_name, "_Heatmap_Top30_DEGs.png")), pheatmap_obj, width = 10, height = 8)
128
129     message(paste(" Analyse für", analysis_name, "abgeschlossen ")); message("------------------------------------------------------------------\n")
130
131     return(list(results_df = res_df, dds = dds, ma = ma_plot, volcano = volcano_plot, pca = pca_plot, umap = umap_plot, heatmap = pheatmap_obj))
132 }
133
134 ################################################################################
135 #  TEIL 2: ANALYSEN AUSFÜHREN
136 ################################################################################
137 analysis_standard <- run_deseq_analysis(standard_counts_path, standard_output_dir, "standard")
138 analysis_manipulated <- run_deseq_analysis(manipulated_counts_path, manipulated_output_dir, "manipulated")
139
140 res_standard <- analysis_standard$results_df; res_manipulated <- analysis_manipulated$results_df
141 dds_std <- analysis_standard$dds
142 ma_plot_std <- analysis_standard$ma; ma_plot_man <- analysis_manipulated$ma
143 volcano_plot_std <- analysis_standard$volcano; volcano_plot_man <- analysis_manipulated$volcano
144 pca_plot_std <- analysis_standard$pca; pca_plot_man <- analysis_manipulated$pca
145 umap_plot_std <- analysis_standard$umap; umap_plot_man <- analysis_manipulated$umap
146 heatmap_std <- analysis_standard$heatmap; heatmap_man <- analysis_manipulated$heatmap
147
148 ################################################################################
149 #  TEIL 3: VERGLEICHSANALYSE
150 ################################################################################
151 message(" Starte Vergleichsanalyse der Robustheit ")
152
153 res_merged <- full_join(res_standard %>% select(Symbol, Ensembl, log2FoldChange, pvalue, padj),
```

```
                        res_manipulated %>% select(Symbol, log2FoldChange, pvalue, padj),
                        by = "Symbol", suffix = c("_std", "_man")) %>%
              filter(!is.na(log2FoldChange_std) & !is.na(log2FoldChange_man))

# Korrelations-Plot
cor_pearson <- cor(res_merged$log2FoldChange_std, res_merged$log2FoldChange_man, method = "pearson")
cor_spearman <- cor(res_merged$log2FoldChange_std, res_merged$log2FoldChange_man, method = "spearman") # Spearman-Berechnung wieder hinzugefügt

lfc_corr_plot <- ggplot(res_merged, aes(x = log2FoldChange_std, y = log2FoldChange_man)) +
    geom_point(alpha = 0.2, color = "blue") +
    geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
    theme_bw() +
    coord_fixed() +
    labs(
        title = "Korrelation der log2 Fold Changes",
        subtitle = paste0("Pearson: ", round(cor_pearson, 3), " | Spearman: ", round(cor_spearman, 3)), # Untertitel mit Werten wiederhergestellt
        caption = bquote(atop(bold("Formel:") ~ r == frac(sum((x[i]-bar(x))(y[i]-bar(y))),sqrt(sum((x[i]-bar(x))^2)*sum((y[i]-bar(y))^2))),
                              bold("Kommentar:") ~ "Misst die lineare Beziehung der Effektstärken. Wert nahe 1 zeigt hohe Stabilität."))
    )
# extra png
ggsave(file.path(final_output_dir, "vergleich_lfc_correlation_plot.png"), lfc_corr_plot, width = 8, height = 8)

# Bland-Altman Plot
bland_altman_df <- res_merged %>% mutate(average_lfc = (log2FoldChange_std + log2FoldChange_man)/2, difference_lfc = log2FoldChange_std - log2FoldChange_man)
mean_diff <- mean(bland_altman_df$difference_lfc, na.rm = TRUE)
sd_diff <- sd(bland_altman_df$difference_lfc, na.rm = TRUE) # Berechnung der Standardabweichung wieder hinzugefügt

bland_altman_plot <- ggplot(bland_altman_df, aes(x = average_lfc, y = difference_lfc)) +
    geom_point(alpha = 0.2) +
    # Alle drei Linien wiederhergestellt
    geom_hline(yintercept = mean_diff, color = "red", linetype = "solid") +
    geom_hline(yintercept = mean_diff + 1.96 * sd_diff, color = "red", linetype = "dashed") +
    geom_hline(yintercept = mean_diff - 1.96 * sd_diff, color = "red", linetype = "dashed") +
    theme_bw() +
    labs(title = "Bland-Altman Plot",
         caption = bquote(atop(bold("Formel:") ~ y == log2FC[std] - log2FC[man],
                               bold("Kommentar:") ~ "Sucht nach systematischem Bias durch Auftragen der Differenz gegen den Mittelwert.")))
# extra png
ggsave(file.path(final_output_dir, "vergleich_bland_altman_plot.png"), bland_altman_plot, width = 8, height = 7)

# P-Wert-Verteilung
pvalue_dist_plot <- ggplot(res_merged) +
    geom_histogram(aes(x = pvalue_std, y = after_stat(density), fill = "Standard"), alpha
```

```
    = 0.6, bins = 50) +
197     geom_histogram(aes(x = pvalue_man, y = after_stat(density), fill = "Manipuliert"), alpha = 0.6, bins = 50) +
198     scale_fill_manual(name = "Datensatz", values = c("Standard" = "blue", "Manipuliert" = "red")) +
199     theme_minimal() + labs(title = "Verteilung der P-Werte",
200                            caption = "Kommentar: Dient der Qualitätskontrolle. Eine flache Verteilung mit einer Spitze bei Null deutet auf eine korrekte statistische Modellierung hin.")
201 # extra png
202 ggsave(file.path(final_output_dir, "vergleich_pvalue_distribution_plot.png"), pvalue_dist_plot, width = 8, height = 6)
203
204 # Venn-Diagramm
205 degs_std <- res_merged %>% filter(padj_std < 0.05 & abs(log2FoldChange_std) > 1) %>% pull(Symbol)
206 degs_man <- res_merged %>% filter(padj_man < 0.05 & abs(log2FoldChange_man) > 1) %>% pull(Symbol)
207 venn_list <- list(Standard = degs_std, Manipuliert = degs_man)
208 venn_plot <- ggvenn(venn_list, fill_color = c("#0073C2FF", "#EFC000FF"), stroke_size = 0.5, set_name_size = 4) +
209     labs(title = "Überschneidung der signifikanten Gene",
210          caption = bquote(atop(bold("Formel:") ~ "Jaccard-Index" == frac("|A" %intersect% "B|", "|A" %union% "B|"),
211                                bold("Kommentar:") ~ "Quantifiziert die Überlappung der Trefferlisten. Ein hoher Index bedeutet hohe Stabilität.")))
212 # extra png
213 ggsave(file.path(final_output_dir, "vergleich_venn_diagram.png"), venn_plot, width = 7, height = 7)
214
215 # GSEA Plot (Funktion und Aufruf)
216 run_gsea <- function(res_df, analysis_name) {
217     gene_list <- res_df$log2FoldChange; names(gene_list) <- res_df$Ensembl; gene_list <- sort(gene_list, decreasing = TRUE)
218     gene_list <- gene_list[!is.na(gene_list) & !is.na(names(gene_list)) & !duplicated(names(gene_list))]
219     if (length(gene_list) == 0) return(NULL)
220     gse <- gseGO(geneList = gene_list, OrgDb = org.Hs.eg.db, keyType = "ENSEMBL", ont = "BP", pAdjustMethod = "BH", pvalueCutoff = 0.1, verbose = FALSE)
221     if (is.null(gse) || nrow(as.data.frame(gse)) == 0) return(NULL); return(gse)
222 }
223 gsea_std <- run_gsea(res_standard, "standard"); gsea_man <- run_gsea(res_manipulated, "manipulated")
224 gsea_comp_plot <- NULL
225 if (!is.null(gsea_std) && !is.null(gsea_man)) {
226     top_std <- as.data.frame(gsea_std) %>% head(10) %>% mutate(Dataset = "Standard")
227     top_man <- as.data.frame(gsea_man) %>% head(10) %>% mutate(Dataset = "Manipulated")
228     gsea_comp_df <- rbind(top_std, top_man)
229     gsea_comp_df$Description <- factor(gsea_comp_df$Description, levels = rev(unique(gsea_comp_df$Description)))
230     gsea_comp_plot <- ggplot(gsea_comp_df, aes(x = enrichmentScore, y = Description, color = p.adjust)) +
231         geom_point(aes(size = setSize)) + facet_grid(. ~ Dataset) + scale_color_gradient(low = "red", high = "blue") +
232         theme_bw() + theme(axis.text.y = element_text(size = 8)) +
```

```
233         labs(title = "GSEA GO Vergleich: Top Angereicherte Pathways",
234              caption = "Kommentar: Prüft, ob Gen-Sets (Pfade) am Anfang/Ende der Gen-Rangliste angereichert sind, um die biologische Interpretation zu validieren.")
235     # extra png
236     ggsave(file.path(final_output_dir, "vergleich_gsea_plot.png"), gsea_comp_plot, width = 10, height = 8)
237 }
238
239 ################################################################################
240 #  TEIL 5: LFC-VERTEILUNG
241 ################################################################################
242 lfc_values <- res_standard$log2FoldChange[!is.na(res_standard$log2FoldChange)]
243 mu <- mean(lfc_values); sigma <- sd(lfc_values)
244 x_curve <- seq(mu - 4 * sigma, mu + 4 * sigma, length.out = 400); y_curve <- dnorm(x_curve, mean = mu, sd = sigma)
245 normal_curve_df <- data.frame(x = x_curve, y = y_curve)
246 norm_vergleich_plot <- ggplot(normal_curve_df, aes(x = x, y = y)) +
247     geom_line(color = "black", linewidth = 1) + geom_vline(xintercept = mu, linetype = "solid", color = "grey40") +
248     annotate("segment", x = mu - sigma, y = max(y_curve)/2, xend = mu + sigma, yend = max(y_curve)/2, arrow = arrow(length = unit(0.2, "cm"), ends = "both")) +
249     annotate("text", x = mu, y = max(y_curve)/1.8, label = "sigma", size = 6, parse = TRUE) +
250     annotate("text", x = mu, y = -0.02, label = "mu", size = 6, parse = TRUE) + theme_minimal(base_size = 14) +
251     labs(title = "Normalverteilung basierend auf Daten", subtitle = bquote(mu == .(round(mu, 3)) ~ "," ~ sigma == .(round(sigma, 3))),
252          caption = bquote(atop(bold("Formel:") ~ f(x) == frac(1, sqrt(2*pi*sigma^2)) * e^(-frac((x-mu)^2, 2*sigma^2)),
253                                bold("Kommentar:") ~ "Zeigt die theoretische Verteilung der log2FC-Werte unter Annahme einer Normalverteilung.")))
254 # extra png
255 ggsave(file.path(final_output_dir, "theoretische_lfc_normalverteilung_plot.png"), norm_vergleich_plot, width = 9, height = 7)
256
257 ################################################################################
258 #  TEIL 6: NEGATIV-VERTEILUNGEN
259 ################################################################################
260 message("\n Erstelle Plots für Binomial- und Negativ-Binomialverteilungen ")
261 # Binomial
262 binom_size <- 20; binom_prob <- 0.3
263 binom_df <- data.frame(x = 0:binom_size, y = dbinom(0:binom_size, size = binom_size, prob = binom_prob))
264 binomial_plot <- ggplot(binom_df, aes(x = x, y = y)) + geom_col(fill = "darkcyan") +
265     theme_minimal(base_size = 14) + labs(title = "Beispiel einer Binomialverteilung", subtitle = paste0("n = ", binom_size, ", p = ", binom_prob),
266          caption = bquote(atop(bold("Formel:") ~ P(X==k) == bgroup("(", atop(n, k), ")") * p^k * (1-p)^(n-k),
267                                bold("Kommentar:") ~ "Beschreibt die Wahrscheinlichkeit von 'k' Erfolgen in 'n' Versuchen (didaktisches Beispiel).")))
268 # extra png
269 ggsave(file.path(final_output_dir, "binomial_verteilung_plot.png"), binomial_plot, width = 8, height = 6)
270
271 # Negativ-Binomial
```

```
272 valid_genes <- !is.na(mcols(dds_std)$dispersion) & mcols(dds_std)$baseMean > 0
273 dispersions <- mcols(dds_std)$dispersion[valid_genes]; base_means <- mcols(dds_std)$↩
    baseMean[valid_genes]
274 target_gene_idx <- which.min(abs(base_means - median(base_means))); target_gene_name <- ↩
    rownames(dds_std)[valid_genes][target_gene_idx]
275 gene_mu <- base_means[target_gene_idx]; gene_disp <- dispersions[target_gene_idx]; gene_↩
    size <- 1 / gene_disp
276 actual_counts <- counts(dds_std, normalized=TRUE)[target_gene_name, ]; counts_df <- data.↩
    frame(counts = actual_counts)
277 x_vals <- 0:round(max(actual_counts) * 1.2); nbinom_df <- data.frame(x = x_vals, y = ↩
    dnbinom(x_vals, size = gene_size, mu = gene_mu))
278 neg_binomial_plot <- ggplot(counts_df, aes(x = counts)) +
279     geom_histogram(aes(y = after_stat(density)), binwidth = 1, fill = "coral", color = "↩
    black", alpha = 0.7) +
280     geom_line(data = nbinom_df, aes(x = x, y = y), color = "blue", linewidth = 1.2) + ↩
    theme_minimal(base_size = 14) +
281     labs(title = "Negative Binomialverteilung (Beispiel für ein Gen)", subtitle = paste0("↩
    Angepasst an Gen '", target_gene_name, "'"),
282          caption = bquote(atop(bold("Formel:") ~ P(X==k) == bgroup("(", atop(k+r-1, k), ")↩
    ") * p^k * (1-p)^r,
283                                bold("Kommentar:") ~ "Das von DESeq2 verwendete Modell, um ↩
    die 'Overdispersion' in RNA-Seq-Daten zu berücksichtigen.")))
284 # extra png
285 ggsave(file.path(final_output_dir, "negativ_binomial_plot.png"), neg_binomial_plot, width ↩
    = 9, height = 7)
286
287 ##############################################################################################
288 #  TEIL 7: VERGLEICH DER NORMALVERTEILUNGEN
289 ##############################################################################################
290
291 message("\n Erstelle Vergleichsplot der Normalverteilungen ")
292
293 # 1. Parameter für Standard-Datensatz extrahieren
294 lfc_std <- res_standard$log2FoldChange[!is.na(res_standard$log2FoldChange)]
295 mu_std <- mean(lfc_std)
296 sigma_std <- sd(lfc_std)
297
298 # 2. Parameter für Manipulierten-Datensatz extrahieren
299 lfc_man <- res_manipulated$log2FoldChange[!is.na(res_manipulated$log2FoldChange)]
300 mu_man <- mean(lfc_man)
301 sigma_man <- sd(lfc_man)
302
303 # Gib die berechneten Werte zur Information aus
304 message(paste(">>> Standard:     mu =", round(mu_std, 4), "| sigma =", round(sigma_std, 4)↩
    ))
305 message(paste(">>> Manipuliert:  mu =", round(mu_man, 4), "| sigma =", round(sigma_man, 4)↩
    ))
306
307 # 3. Erstelle einen gemeinsamen Datenrahmen für beide Verteilungskurven
308 x_range <- seq(min(mu_std - 4 * sigma_std, mu_man - 4 * sigma_man),
309                max(mu_std + 4 * sigma_std, mu_man + 4 * sigma_man),
310                length.out = 500)
311
312 curve_std_df <- data.frame(x = x_range, y = dnorm(x_range, mu_std, sigma_std), dataset = "↩
    Standard")
```

```
curve_man_df <- data.frame(x = x_range, y = dnorm(x_range, mu_man, sigma_man), dataset = "Manipuliert")
curves_df <- rbind(curve_std_df, curve_man_df)

# 4. Erstelle den finalen Vergleichsplot
norm_dist_comparison_plot <- ggplot(curves_df, aes(x = x, y = y, color = dataset)) +
    geom_line(linewidth = 1.2) +
    geom_vline(xintercept = mu_std, color = "#0073C2FF", linetype = "dashed") +
    geom_vline(xintercept = mu_man, color = "#EFC000FF", linetype = "dashed") +
    scale_color_manual(values = c("Standard" = "#0073C2FF", "Manipuliert" = "#EFC000FF")) +
    labs(
        title = "Vergleich der Normalverteilungen der log2 Fold Changes",
        subtitle = bquote("Standard (blau):" ~ mu == .(round(mu_std, 3)) * "," ~ sigma == .(round(sigma_std, 3)) ~
                          "|" ~ "Manipuliert (gelb):" ~ mu == .(round(mu_man, 3)) * "," ~ sigma == .(round(sigma_man, 3))),
        x = "log2 Fold Change",
        y = "Dichte",
        caption = "Kommentar: Vergleicht die Gesamtverteilung der Effektstärken. Ähnliche Kurven deuten auf eine hohe strukturelle Stabilität hin."
    ) +
    theme_minimal(base_size = 14)

ggsave(file.path(final_output_dir, "norm_dist_comparison_plot.png"), norm_dist_comparison_plot, width = 9, height = 7)
message("Vergleichsplot der Normalverteilungen gespeichert.")

################################################################################
#  TEIL 8: VERGLEICH DER ECHTEN DATENVERTEILUNGEN
################################################################################

message("\n Erstelle Vergleichsplot der echten Dichteverteilungen ")

# 1. Erstelle einen kombinierten Datenrahmen mit den log2FC-Werten beider Analysen
lfc_std_df <- data.frame(
    log2FoldChange = res_standard$log2FoldChange[!is.na(res_standard$log2FoldChange)],
    dataset = "Standard"
)
lfc_man_df <- data.frame(
    log2FoldChange = res_manipulated$log2FoldChange[!is.na(res_manipulated$log2FoldChange)],
    dataset = "Manipuliert"
)
lfc_combined_df <- rbind(lfc_std_df, lfc_man_df)

# Berechne die Mittelwerte für die vertikalen Linien
mu_std <- mean(lfc_std_df$log2FoldChange)
mu_man <- mean(lfc_man_df$log2FoldChange)

# 2. Erstelle den Vergleichsplot der Dichtekurven
density_comparison_plot <- ggplot(lfc_combined_df, aes(x = log2FoldChange, color = dataset)) +
    geom_density(linewidth = 1.2) +
    geom_vline(xintercept = mu_std, color = "#0073C2FF", linetype = "dashed") +
```

```
360     geom_vline(xintercept = mu_man, color = "#EFC000FF", linetype = "dashed") +
361     scale_color_manual(values = c("Standard" = "#0073C2FF", "Manipuliert" = "#EFC000FF")) +
362     labs(
363         title = "Vergleich der Dichteverteilungen der log2 Fold Changes",
364         subtitle = "Vergleich der empirischen (echten) Datenverteilungen",
365         x = "log2 Fold Change",
366         y = "Dichte",
367         caption = bquote(atop(bold("Methode:") ~ "Kernel-Dichte-Schätzung (KDE) aus den echten Datenpunkten.",
368                               bold("Kommentar:") ~ "Fast identische Kurven belegen eine hohe strukturelle Stabilität der Ergebnisse."))
369     ) +
370     theme_minimal(base_size = 14)
371
372 ggsave(file.path(final_output_dir, "density_comparison_plot.png"), density_comparison_plot, width = 9, height = 7)
373 message("Vergleichsplot der Dichteverteilungen gespeichert.")
374
375 ################################################################################
376 #  TEIL 9: SAMMEL-PDF ERSTELLEN
377 ################################################################################
378
379 message("\n Erstelle Sammel-PDF mit allen Analyse- und Vergleichs-Plots ")
380 pdf_filename <- file.path(final_output_dir, "final_summary_report.pdf")
381
382 # Finale, geordnete Liste aller Plots
383 plots_to_include <- list(
384     "MA-Plot (Standard)" = ma_plot_std, "Volcano Plot (Standard)" = volcano_plot_std,
385     "PCA Plot (Standard)" = pca_plot_std, # "UMAP Plot (Standard)" = umap_plot_std,
386     "Heatmap (Standard)" = heatmap_std, "MA-Plot (Manipuliert)" = ma_plot_man,
387     "Volcano Plot (Manipuliert)" = volcano_plot_man, "PCA Plot (Manipuliert)" = pca_plot_man,
388     # "UMAP Plot (Manipuliert)" = umap_plot_man,
389     "Heatmap (Manipuliert)" = heatmap_man,
390     "Korrelation der log2FC" = lfc_corr_plot, "Bland-Altman Plot" = bland_altman_plot,
391     "P-Wert Verteilung" = pvalue_dist_plot, "DEG Überlappung" = venn_plot,
392     "GSEA Vergleich" = gsea_comp_plot, "Normalverteilung der LFC (Standard)" = norm_vergleich_plot,
393     # "Vergleich Dichteverteilungen (Echte Daten)" = density_comparison_plot, # NEU HINZUGEFÜGT
394     "Vergleich der Normalverteilungen" = norm_dist_comparison_plot #, "Beispiel Binomialverteilung" = binomial_plot, "Beispiel Negativ-Binomialverteilung" = neg_binomial_plot
395 )
396
397 tryCatch({
398     pdf(pdf_filename, width = 12, height = 9)
399     for (plot_name in names(plots_to_include)) {
400         if (is.null(plots_to_include[[plot_name]])) next
401         message("Füge Plot '", plot_name, "' zum PDF hinzu...")
402         pl <- plots_to_include[[plot_name]]
403
404         if (inherits(pl, "ggplot")) { print(pl) }
405         else if (inherits(pl, "pheatmap")) { grid::grid.newpage(); grid::grid.draw(pl$
```

```
    gtable) }
406     }
407 }, error = function(e) {
408     warning(paste("Fehler beim Erstellen des PDFs:", e$message))
409 }, finally = {
410     if(dev.cur() != 1) dev.off()
411     if (file.exists(pdf_filename)) { message("ERFOLG: Sammel-PDF mit Plots erstellt: ", ↩
    pdf_filename) }
412     else { warning("WARNUNG: Sammel-PDF konnte nicht erstellt werden: ", pdf_filename) }
413 })
414
415 message("\n--------------------------------------------------------------------------")
416 message("Alle Analysen abgeschlossen. Ergebnisse sind in 'output_final/'.")
417 message("--------------------------------------------------------------------------")
418 # vim: set fdm=marker:
```

Literaturverzeichnis

[1] M. Ashburner u. a. „Gene ontology: tool for the unification of biology“. In: *Nat. Genet.* 25.1 (2000), S. 25–29. DOI: 10.1038/75556.

[2] G. Baruzzo u. a. „Simulation-based comprehensive benchmarking of RNA-seq aligners“. In: *Nature Methods* 14.2 (2017), S. 135–139. DOI: 10.1038/nmeth.4106.

[3] Y. Benjamini und Y. Hochberg. „Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing“. In: *J. R. Stat. Soc. Series B Stat. Methodol.* 57.1 (1995), S. 289–300. DOI: 10.1111/j.2517-6161.1995.tb02031.x.

[4] H. J. C. Berendsen. „Errors: classification and propagation“. In: *A Student's Guide to Data and Error Analysis*. Cambridge: Cambridge University Press, 2011. Kap. 3, S. 18–26. DOI: 10.1017/CBO9780511921247.004.

[5] *Bernoulli-Verteilung*. Wikipedia — Die freie Enzyklopädie, https://de.wikipedia.org/wiki/Bernoulli-Verteilung. Zugriff am 27. Sep 2025. 2025.

[6] J. M. Bland und D. G. Altman. „Statistical methods for assessing agreement between two methods of clinical measurement“. In: *Lancet* 1.8476 (1986), S. 307–310. DOI: 10.1016/S0140-6736(86)90837-8.

[7] N. L. Bray u. a. „Near-optimal probabilistic RNA-seq quantification“. In: *Nat. Biotechnol.* 34.5 (2016), S. 525–527. DOI: 10.1038/nbt.3519.

[8] Ana Conesa u. a. „A survey of best practices for RNA-seq data analysis“. In: *Genome Biol.* 17 (2016), S. 13. DOI: 10.1186/s13059-016-0881-8.

[9] A. Der Kiureghian und O. Ditlevsen. „Aleatory or epistemic? Does it matter?“ In: *Struct. Saf.* 31.2 (2009), S. 105–112. DOI: 10.1016/j.strusafe.2008.06.020.

[10] A. Dobin u. a. „STAR: ultrafast universal RNA-seq aligner“. In: *Bioinformatics* 29.1 (2013), S. 15–21. DOI: 10.1093/bioinformatics/bts635.

[11] P. G. Engström u. a. „Systematic evaluation of spliced alignment programs for RNA-seq data“. In: *Nat. Methods* 10.12 (2013), S. 1185–1191. DOI: 10.1038/nmeth.2722.

[12] *GSE52778: Human Airway Smooth Muscle Transcriptome Changes in Response to Asthma Medications*. NCBI Gene Expression Omnibus (GEO), https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE52778. Zugriff am 26. Sep 2025. 2014.

[13] S. A. Hardwick u. a. „Spliced synthetic genes as internal controls in RNA sequencing experiments“. In: *Nat. Methods* 13 (2016), S. 792–798. DOI: 10.1038/nmeth.3953.

[14] B. E. Himes u. a. „RNA-Seq transcriptome profiling identifies CRISPLD2 as a glucocorticoid responsive gene that modulates cytokine function in airway smooth muscle cells“. In: *PLoS One* 9.6 (2014), e99625. DOI: 10.1371/journal.pone.0099625.

[15] Are Hjørungnes. *Complex-Valued Matrix Derivatives: With Applications in Signal Processing and Communications*. Cambridge University Press, 2011. ISBN: 9780521192644. DOI: 10.1017/CBO9780511921490.

[16] JCGM. *Evaluation of measurement data – Guide to the expression of uncertainty in measurement*. Techn. Ber. Joint Committee for Guides in Metrology (BIPM), 2008. URL: https://www.bipm.org/documents/20126/2071204/JCGM_100_2008_E.pdf.

[17] Lichun Jiang u. a. „Synthetic spike-in standards for RNA-seq experiments“. In: *Genome Res.* 21.9 (2011), S. 1543–1551. DOI: 10.1101/gr.121095.111.

[18] F. Krueger. *Trim Galore!* https://www.bioinformatics.babraham.ac.uk/projects/trim_galore/. 2012.

[19] H. H. Ku. „Notes on the Use of Propagation of Error Formulas“. In: *J. Res. Natl. Bur. Stand. C Eng. Instrum.* 70C.4 (1966), S. 263–273. DOI: 10.6028/jres.070C.025.

[20] H. Li u. a. „The Sequence Alignment/Map format and SAMtools“. In: *Bioinformatics* 25.16 (2009), S. 2078–2079. DOI: 10.1093/bioinformatics/btp352.

[21] Y. Liao, G. K. Smyth und W. Shi. „featureCounts: an efficient general purpose program for assigning sequence reads to genomic features“. In: *Bioinformatics* 30.7 (2014), S. 923–930. DOI: 10.1093/bioinformatics/btt656.

[22] M. I. Love, W. Huber und S. Anders. „Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2“. In: *Genome Biol.* 15.12 (2014), S. 550. DOI: 10.1186/s13059-014-0550-8.

[23] Thomas Ludwig. *Vorlesungsfolien: Betriebssystemaspekte*. https://wr.informatik.uni-hamburg.de/teaching/wintersemester_2023_2024/hochleistungsrechnen. Modul: Hochleistungsrechnen (WS 2023/2024), Universität Hamburg. Prof. Dr. Thomas Ludwig. Universität Hamburg, 2024.

[24] Thomas Ludwig. *Vorlesungsfolien: Parallele Programmierung*. https://wr.informatik.uni-hamburg.de/teaching/wintersemester_2023_2024/hochleistungsrechnen. Modul: Hochleistungsrechnen (WS 2023/2024), Universität Hamburg. Prof. Dr. Thomas Ludwig. Universität Hamburg, 2024.

[25] Jan R. Magnus und Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. 3. Aufl. John Wiley & Sons, 2019. ISBN: 9781119541202. DOI: 10.1002/9781119541219.

[26] F. J. Martin u. a. „Ensembl 2022“. In: *Nucleic Acids Res.* 50.D1 (2022), S. D988–D995. DOI: 10.1093/nar/gkab1049.

[27] *Monte-Carlo-Simulation*. Wikipedia – Die freie Enzyklopädie, https://de.wikipedia.org/wiki/Monte-Carlo-Simulation. Zugriff am 27. Sep 2025. 2025.

[28] G. W. Oehlert. „A Note on the Delta Method“. In: *Am. Stat.* 46.1 (1992), S. 27–29. DOI: 10.1080/00031305.1992.10475842.

[29] R. Patro u. a. „Salmon provides fast and bias-aware quantification of transcript expression“. In: *Nat. Methods* 14.4 (2017), S. 417–419. DOI: 10.1038/nmeth.4197.

[30] Kaare Brandt Petersen und Michael Syskind Pedersen. *The Matrix Cookbook*. Version 20121115. Technical University of Denmark. Lyngby, 2012. URL: https://www2.compute.dtu.dk/pubdb/edoc/imm3274.pdf.

[31] F. Pfeiffer u. a. „Systematic evaluation of error rates and causes in short-read next-generation sequencing“. In: *Sci. Rep.* 8.1 (2018), S. 10950. DOI: 10.1038/s41598-018-29325-6.

[32] *PRJNA229998: BioProject / SRA for GSE52778 (raw sequencing runs)*. NCBI BioProject / SRA, https://www.ncbi.nlm.nih.gov/bioproject/PRJNA229998 (oder Traces view: https://www.ncbi.nlm.nih.gov/Traces/study/?acc=PRJNA229998). Zugriff am 26. Sep 2025. 2014.

[33] Chet Ramey und Brian Fox. *GNU Bash Reference Manual*. Abschnitte: Signals (trap), Shell Parameters ($RANDOM). Free Software Foundation. 2025. URL: https://www.gnu.org/software/bash/manual/bash.pdf (besucht am 30. 09. 2025).

[34] C. Robert und M. Watson. „Errors in RNA-Seq quantification affect genes of relevance to human disease“. In: *Genome Biol.* 16 (2015), S. 177. DOI: 10.1186/s13059-015-0734-x.

[35] N. J. Schurch u. a. „How many biological replicates are needed in an RNA-seq experiment and which differential expression tool should you use?“ In: *RNA* 22.6 (2016), S. 839–851. DOI: 10.1261/rna.053959.115.

[36] SEQC/MAQC-III Consortium. „A comprehensive assessment of RNA-seq accuracy, reproducibility and information content“. In: *Nat. Biotechnol.* 32.9 (2014), S. 903–914. DOI: 10.1038/nbt.2957.

[37] *Spike-In RNA Variant (SIRV) Control Mixes*. Lexogen product information and user guides. 2016–2021. URL: https://www.lexogen.com/sirvs/.

[38] SRA Toolkit Development Team. *SRA Toolkit*. https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=software. Accessed: 2025-08-23. 2024.

[39] A. Srivastava u. a. „Alignment and mapping methodology influence transcript abundance estimation“. In: *Genome Biol.* 21 (2020), S. 239. DOI: 10.1186/s13059-020-02151-8.

[40] R. Stark, M. B. Clark und I. Papatheodorou. „RNA sequencing: the teenage years“. In: *Nat. Rev. Genet.* 20.11 (2019), S. 631–656. DOI: 10.1038/s41576-019-0150-2.

[41] A. Subramanian u. a. „Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles“. In: *Proc. Natl. Acad. Sci. U.S.A.* 102.43 (2005), S. 15545–15550. DOI: 10.1073/pnas.0506580102.

[42] Barry N. Taylor und Chris E. Kuyatt. *Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results*. Techn. Ber. Gaithersburg, MD: National Institute of Standards und Technology, 1994. DOI: 10.6028/NIST.TN.1297.

[43] Z. Wang, M. Gerstein und M. Snyder. „RNA-Seq: a revolutionary tool for transcriptomics“. In: *Nat. Rev. Genet.* 10.1 (2009), S. 57–63. DOI: 10.1038/nrg2484.

[44] G. Yu u. a. „clusterProfiler: an R package for comparing biological themes among gene clusters“. In: *OMICS* 16.5 (2012), S. 284–287. DOI: 10.1089/omi.2011.0118.

Eidesstattliche Erklärung

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Bioinformatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe. Sofern im Zuge der Erstellung der vorliegenden Abschlussarbeit generative Künstliche Intelligenz (gKI) basierte elektronische Hilfsmittel verwendet wurden, versichere ich, dass meine eigene Leistung im Vordergrund stand und dass eine vollständige Dokumentation aller verwendeten Hilfsmittel gemäß der Guten Wissenschaftlichen Praxis vorliegt. Ich trage die Verantwortung für eventuell durch die gKI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate.

Hamburg, den 08.10.2025

Vorname Nachname