# Rico V5 Autopilot - Experimente Guide

## Übersicht

Dieser Guide erklärt, wie man A/B Tests und Experimente mit dem Rico V5 Autopilot durchführt. Experimente sind der Kern der selbstverbessernden Funktionalität und ermöglichen kontinuierliche Optimierung.

## Experiment-Typen

### 1. A/B Tests

**Zweck**: Vergleich zwischen zwei Varianten **Beispiele**:

- Prompt-Varianten
- Provider-Konfigurationen
- UI-Änderungen

**Konfiguration:**

```
{
  "name": "Prompt Clarity Test",
  "type": "ab",
  "variants": {
    "A": "Original prompt",
    "B": "Improved prompt with examples"
  },
  "traffic_split": {
    "A": 0.5,
    "B": 0.5
  }
}
```

## 2. Prompt-Optimierung

**Zweck**: Optimierung von AI-Prompts **Beispiele**:

- Klarheits-Verbesserungen
- Constraint-Hinzufügung
- Few-Shot-Beispiele

**Konfiguration:**

```
{
  "name": "Prompt Optimization",
  "type": "prompt",
  "variants": {
    "A": "Original prompt",
    "B": "Optimized prompt with constraints"
  },
  "traffic_split": {
    "A": 0.5,
    "B": 0.5
  }
}
```

## 3. Routing-Optimierung

**Zweck**: Optimierung der Provider-Auswahl **Beispiele**:

- Gewichtungs-Anpassungen
- Latenz-Optimierung
- Kosten-Optimierung

**Konfiguration:**

```
{
  "name": "Routing Optimization",
  "type": "routing",
  "variants": {
    "A": "Current routing",
    "B": "Optimized routing"
  },
```

```
  "traffic_split": {
    "A": 0.5,
    "B": 0.5
  }
}
```

## Experiment-Lebenszyklus

### 1. Erstellung

```
curl -X POST http://localhost:8000/v2/autopilot/experiments \
  -H "Content-Type: application/json" \
  -d '{
    "name": "My Experiment",
    "type": "ab",
    "variants": {
      "A": "Control",
      "B": "Treatment"
    },
    "traffic_split": {
      "A": 0.5,
      "B": 0.5
    },
    "duration_hours": 24,
    "min_samples": 100
  }'
```

### 2. Start

```
curl -X POST http://localhost:8000/v2/autopilot/experiments/exp_123/star
```

### 3. Überwachung

```
# Status prüfen
curl -X GET http://localhost:8000/v2/autopilot/experiments/exp_123/statu

# Auswertung
curl -X POST http://localhost:8000/v2/autopilot/experiments/exp_123/eval
```

### 4. Stopp

```
curl -X POST http://localhost:8000/v2/autopilot/experiments/exp_123/stop
```

## Traffic-Splitting

### Deterministisches Splitting

Das System verwendet deterministisches Splitting basierend auf der `run_id`:

```
# Pseudocode
hash_value = hash(run_id) % 10000
if hash_value < 5000:
    variant = "A"
else:
    variant = "B"
```

### Sticky Sessions

- Gleiche `run_id` → gleiche Variante
- Konsistente User-Experience
- Vermeidung von Varianten-Wechseln

### Traffic-Anpassung

```
# Traffic-Split ändern
curl -X POST http://localhost:8000/v2/autopilot/experiments/exp_123/traf
  -H "Content-Type: application/json" \
  -d '{
    "A": 0.3,
    "B": 0.7
  }'
```

## Erfolgs-Kriterien

### Statistische Signifikanz

- **p-Wert**: < 0.05
- **Konfidenz**: > 80%
- **Sample-Größe**: Mindestens 100 pro Variante

### Business-Kriterien

- **Win-Rate-Delta**: Mindestens 5% Verbesserung
- **Qualität**: Keine Verschlechterung
- **Latenz**: Akzeptable Performance

### Beispiel-Konfiguration

```
{
  "success_criteria": {
    "win_rate_delta_min": 0.05,
    "p_value_max": 0.05,
    "min_confidence": 0.8,
    "min_samples": 100
  }
}
```

## Guardrails

### Kosten-Limits

```
{
  "guardrails": {
    "max_cost_per_day": 10.0,
    "max_cost_per_request": 0.05
  }
}
```

### Qualitäts-Limits

```
{
  "guardrails": {
    "min_quality_score": 0.6,
    "max_error_rate": 0.1
  }
}
```

### Performance-Limits

```
{
  "guardrails": {
    "max_latency_ms": 15000,
    "min_throughput": 10
  }
}
```

# Automatische Auswertung

## Tägliche Evaluation

Das System führt täglich automatische Auswertungen durch:

1. **Daten-Sammlung**: Metriken der letzten 24h
2. **Statistische Analyse**: Wilson-Score, t-Test
3. **Entscheidung**: Continue/Stop/Apply
4. **Aktion**: Automatische Promotion oder Rollback

## Auswertungs-Logik

```
def evaluate_experiment(experiment):
    # Hole Metriken
    metrics_a = get_metrics(experiment, "A")
    metrics_b = get_metrics(experiment, "B")

    # Statistische Tests
    p_value = wilson_score_test(metrics_a, metrics_b)
    effect_size = calculate_effect_size(metrics_a, metrics_b)

    # Entscheidung
    if p_value < 0.05 and effect_size > 0.05:
        if metrics_b.win_rate > metrics_a.win_rate:
            return "apply_b"
        else:
            return "apply_a"
    else:
        return "continue"
```

# Best Practices

## 1. Experiment-Design

- **Hypothese**: Klare Erwartung formulieren
- **Metriken**: Relevante KPIs definieren
- **Dauer**: Ausreichend lange für Signifikanz

- **Sample-Größe**: Mindestens 100 pro Variante

### 2. Varianten-Erstellung

- **Kleine Änderungen**: Ein Faktor pro Experiment
- **Klarheit**: Verständliche Unterschiede
- **Konsistenz**: Gleiche Struktur und Länge

### 3. Überwachung

- **Tägliche Checks**: Status und Metriken
- **Alerts**: Bei Guardrail-Verletzungen
- **Logs**: Detaillierte Aufzeichnung

### 4. Auswertung

- **Statistische Signifikanz**: $p < 0.05$
- **Praktische Relevanz**: Mindestens 5% Verbesserung
- **Konsistenz**: Über mehrere Zeiträume

## Häufige Probleme

### 1. Keine Signifikanz

**Ursachen:**

- Zu kleine Sample-Größe
- Zu kurze Laufzeit
- Zu kleine Unterschiede

**Lösungen:**

- Längere Laufzeit
- Größere Unterschiede
- Mehr Traffic

### 2. Guardrail-Verletzungen

**Ursachen:**

- Zu hohe Kosten
- Zu schlechte Qualität
- Zu hohe Latenz

**Lösungen:**

- Limits anpassen
- Varianten verbessern
- Experiment stoppen

### 3. Inkonsistente Ergebnisse

**Ursachen:**

- Zu kurze Laufzeit
- Saisonale Effekte
- Zufällige Schwankungen

**Lösungen:**

- Längere Laufzeit
- Mehrfache Wiederholung
- Robustere Metriken

## Monitoring & Alerts

### Dashboard-Metriken

- **Laufende Experimente**: Anzahl und Status
- **Traffic-Split**: Aktuelle Verteilung
- **Win-Rate**: Erfolgsrate pro Variante
- **Latenz**: Durchschnittliche Antwortzeit
- **Kosten**: Tägliche Ausgaben

### Slack-Notifications

- **Experiment-Start**: Benachrichtigung bei Start
- **Signifikante Ergebnisse**: Bei $p < 0.05$
- **Guardrail-Verletzungen**: Bei Limit-Überschreitung
- **Automatische Aktionen**: Bei Promotion/Rollback

### Logs

```
# Experiment-Logs
tail -f logs/backend.log | grep "experiment"

# Metriken-Logs
tail -f logs/backend.log | grep "metrics"

# Scheduler-Logs
tail -f logs/backend.log | grep "scheduler"
```

## Erweiterte Funktionen

### Multi-Variant-Tests

```
{
  "variants": {
    "A": "Control",
    "B": "Treatment 1",
    "C": "Treatment 2"
  },
  "traffic_split": {
    "A": 0.33,
    "B": 0.33,
    "C": 0.34
  }
}
```

### Segmentierte Tests

```
{
  "segments": {
    "new_users": {
      "traffic_split": {"A": 0.5, "B": 0.5}
    },
    "returning_users": {
      "traffic_split": {"A": 0.7, "B": 0.3}
    }
  }
}
```

### Kontinuierliche Tests

- **Rolling Experiments**: Neue Varianten automatisch
- **Auto-Optimization**: Kontinuierliche Verbesserung
- **Adaptive Splitting**: Dynamische Anpassung

## Troubleshooting

### Experiment startet nicht

1. **Traffic-Split prüfen**: Summe muss 1.0 ergeben
2. **Varianten prüfen**: Nicht leer, gültige JSON
3. **Guardrails prüfen**: Limits nicht zu restriktiv
4. **Logs prüfen**: Fehlermeldungen analysieren

### Keine Metriken

1. **Run-ID prüfen**: Korrekte Verwendung
2. **Experiment-ID prüfen**: Korrekte Zuordnung
3. **Provider prüfen**: Unterstützte Provider
4. **API prüfen**: Endpoints erreichbar

### Inkonsistente Ergebnisse

1. **Sample-Größe prüfen**: Mindestens 100 pro Variante
2. **Laufzeit prüfen**: Mindestens 24 Stunden
3. **Traffic prüfen**: Gleichmäßige Verteilung
4. **Metriken prüfen**: Korrekte Erfassung

## Beispiele

### Prompt-Optimierung

```
# 1. Experiment erstellen
curl -X POST http://localhost:8000/v2/autopilot/experiments \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Prompt Clarity Test",
    "type": "prompt",
    "variants": {
      "A": "Du bist ein hilfreicher Assistent.",
      "B": "Du bist ein hilfreicher Assistent mit jahrelanger Erfahrung.
    },
    "traffic_split": {"A": 0.5, "B": 0.5},
    "duration_hours": 48,
    "min_samples": 200
  }'

# 2. Experiment starten
curl -X POST http://localhost:8000/v2/autopilot/experiments/exp_123/star

# 3. Metriken loggen (automatisch über API-Hooks)
# 4. Auswertung (automatisch täglich)
# 5. Promotion (automatisch bei Erfolg)
```

### Provider-Optimierung

```
# 1. Routing-Experiment erstellen
curl -X POST http://localhost:8000/v2/autopilot/experiments \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "name": "Provider Optimization",
    "type": "routing",
    "variants": {
      "A": "Current weights: OpenAI 60%, Claude 40%",
      "B": "Optimized weights: OpenAI 70%, Claude 30%"
    },
    "traffic_split": {"A": 0.5, "B": 0.5}
  }'
```

---

**Rico V5 Autopilot Experiments** - Vollständiger Guide für A/B Tests und kontinuierliche Optimierung