Rico V5 Autopilot API - Referenz

Übersicht

Die Autopilot API bietet REST-Endpoints für die Verwaltung der selbstverbessernden Orchestrierung. Alle Endpoints sind unter `/v2/autopilot/` verfügbar.

Base URL

```
http://localhost:8000/v2/autopilot
```

Authentifizierung

Aktuell keine Authentifizierung erforderlich. In Produktionsumgebungen sollte API-Key-Authentifizierung implementiert werden.

Status & Health

GET /status

Gibt den aktuellen Autopilot-Status zurück.

Response:

```json
{
  "enabled": true,
  "timestamp": "2024-01-01T12:00:00Z",
  "scheduler": {
    "running": true,
    "jobs_count": 4,
      "enabled_jobs": 4
    },
    "experiments": {
      "running_count": 2,
      "experiments": [
        {
          "experiment_id": "exp_123",
          "status": "running",
          "start_time": "2024-01-01T10:00:00Z"
        }
      ]
    },
    "knowledge_base": {
      "total_sources": 15,
      "total_chunks": 1200,
      "total_summaries": 8
    },
    "last_ingest": "2024-01-01T11:00:00Z"
  }
```

GET /health

Gibt detaillierte Health-Informationen zurück.

Response:

```
{
  "overall_status": "healthy",
  "timestamp": "2024-01-01T12:00:00Z",
  "components": {
    "metrics": {
      "status": "ok",
      "message": "Metrics writer available"
    },
    "experiments": {
      "status": "ok",
      "message": "Experiment manager available"
    },
    "knowledge": {
      "status": "ok",
      "message": "Knowledge base available"
    },
    "registry": {
      "status": "ok",
      "message": "Registry available"
    },
    "scheduler": {
        "status": "ok",
        "message": "Scheduler available"
      }
    }
  }
```

Metriken

POST /metrics

Loggt Metriken für einen Run.

Request Body:

```
{
  "task": "ai_ask",
  "provider": "openai",
  "latency_ms": 1500.0,
  "cost_est": 0.01,
  "quality_score": 0.85,
  "win": true,
  "error_type": null,
  "run_id": "run_123",
  "experiment_id": "exp_123",
  "metadata": {
    "model": "gpt-4",
    "tokens": 150
  }
}
```

Response:

```
{
  "status": "success",
  "run_id": "run_123"
}
```

POST /metrics/rollup

Führt Metriken-Rollup durch.

Response:

```json
{
  "status": "success",
  "timestamp": "2024-01-01T12:00:00Z",
  "rollups": {
    "hourly": {
      "total_runs": 50,
      "avg_latency_ms": 1200,
      "total_cost": 0.5,
      "error_rate": 0.02,
      "win_rate": 0.8,
      "avg_quality_score": 0.85
    },
    "daily": {
      "total_runs": 1200,
      "avg_latency_ms": 1100,
      "total_cost": 12.0,
      "error_rate": 0.03,
      "win_rate": 0.82,
      "avg_quality_score": 0.87
    },
    "weekly": {
      "total_runs": 8400,
      "avg_latency_ms": 1050,
      "total_cost": 84.0,
      "error_rate": 0.025,
      "win_rate": 0.84,
      "avg_quality_score": 0.89
    }
  }
}
```

GET /metrics/summary

Gibt Metriken-Zusammenfassung zurück.

Query Parameters:

- `hours` (optional): Stunden zurück (default: 24)
- `task` (optional): Filter nach Task

- `provider` (optional): Filter nach Provider

Response:

```
{
  "status": "success",
  "summary": {
    "total_runs": 1200,
    "avg_latency_ms": 1100,
    "total_cost": 12.0,
    "error_rate": 0.03,
    "win_rate": 0.82,
    "avg_quality_score": 0.87
  },
  "filters": {
    "hours": 24,
    "task": null,
    "provider": null
  }
}
```

Experimente

GET /experiments

Listet Experimente auf.

Query Parameters:

- `status` (optional): Filter nach Status
- `limit` (optional): Maximale Anzahl (default: 50)

Response:

```
{
  "status": "success",
  "experiments": [
    {
      "experiment_id": "exp_123",
      "status": "running",
      "start_time": "2024-01-01T10:00:00Z",
        "end_time": null,
        "current_traffic": {
          "A": 0.5,
          "B": 0.5
        }
      }
    ]
  }
```

POST /experiments

Erstellt neues Experiment.

Request Body:

```
{
  "name": "Prompt Clarity Test",
  "type": "prompt",
  "variants": {
    "A": "Original prompt",
    "B": "Improved prompt with examples"
  },
  "traffic_split": {
    "A": 0.5,
    "B": 0.5
  },
  "duration_hours": 24,
  "min_samples": 100,
  "success_criteria": {
    "win_rate_delta_min": 0.05,
    "p_value_max": 0.05,
    "min_confidence": 0.8
  },
  "guardrails": {
    "max_error_rate": 0.1,
    "max_latency_ms": 15000,
    "max_cost_per_day": 10.0
  }
}
```

Response:

```
{
  "status": "success",
  "experiment_id": "exp_123",
  "message": "Experiment 'Prompt Clarity Test' created successfully"
}
```

POST /experiments/{id}/start

Startet Experiment.

Response:

```
{
  "status": "success",
  "message": "Experiment exp_123 started"
}
```

POST /experiments/{id}/stop

Stoppt Experiment.

Response:

```
{
  "status": "success",
  "message": "Experiment exp_123 stopped"
}
```

GET /experiments/{id}/status

Gibt Status eines Experiments zurück.

Response:

```json
{
  "status": "success",
  "experiment": {
    "experiment_id": "exp_123",
    "status": "running",
    "start_time": "2024-01-01T10:00:00Z",
    "end_time": null,
    "current_traffic": {
      "A": 0.5,
      "B": 0.5
    }
  }
}
```

POST /experiments/{id}/evaluate

Wertet Experiment aus.

Response:

```json
{
  "status": "success",
  "evaluation": {
    "experiment_id": "exp_123",
    "variant_a": "A",
    "variant_b": "B",
    "n_a": 100,
    "n_b": 100,
    "win_rate_a": 0.6,
    "win_rate_b": 0.8,
    "p_value": 0.03,
    "significant": true,
    "effect_size": 0.2,
    "recommendation": "apply_b"
  }
}
```

Evaluation & Optimization

POST /evaluate

Führt System-Evaluation durch.

Response:

```json
{
  "status": "success",
  "evaluation": {
    "timestamp": "2024-01-01T12:00:00Z",
    "total_experiments": 2,
    "experiments": [
      {
        "experiment_id": "exp_123",
        "status": "running",
        "guardrails": {
          "status": "ok",
          "violations": []
        },
        "evaluation": {
          "significant": true,
          "recommendation": "apply_b"
        }
      }
    ],
    "summary": {
      "violations": 0,
      "ready_for_evaluation": 1,
      "auto_stopped": 0
    }
  }
}
```

POST /optimize

Optimiert System.

Request Body:

```
{
  "base_prompt": "Du bist ein hilfreicher Assistent.",
  "objectives": {
    "quality": 0.4,
    "latency": 0.2,
    "cost": 0.2,
    "reliability": 0.2
  }
}
```

Response:

```
{
  "status": "success",
  "optimization": {
    "prompt_variants": [
      {
        "id": "variant_1",
        "name": "Optimized Clarity v1",
        "content": "Du bist ein hilfreicher Assistent mit jahrelanger Er
        "role": "system",
        "tags": ["clarity_improvement", "optimized"]
      }
    ],
    "routing_policy": {
      "id": "policy_1",
      "name": "Optimized Routing 2024-01-01 12:00",
      "weights": {
        "openai": 0.6,
        "claude": 0.4
      },
      "conditions": {
        "max_latency_ms": 10000,
        "max_cost_per_request": 0.05,
        "min_quality_threshold": 0.6
      }
    }
  }
}
```

Registry Management

POST /propose

Schlägt Änderungen vor.

Request Body:

```json
{
  "prompt_variants": [
    {
      "id": "prompt_1",
      "name": "Improved Assistant Prompt",
      "content": "Du bist ein hilfreicher Assistent...",
      "role": "system",
      "tags": ["improved", "clarity"]
    }
  ],
  "routing_policies": [
    {
      "id": "policy_1",
      "name": "Optimized Routing",
      "weights": {
        "openai": 0.7,
        "claude": 0.3
      },
      "conditions": {
        "max_latency_ms": 8000
      }
    }
  ]
}
```

Response:

```json
{
  "status": "success",
  "proposed": {
    "prompts": [
      {
        "id": "prompt_1",
        "name": "Improved Assistant Prompt"
      }
    ],
    "policies": [
      {
        "id": "policy_1",
        "name": "Optimized Routing"
      }
    ]
  }
}
```

POST /apply

Wendet Änderungen an.

Request Body:

```
{
  "prompt_ids": ["prompt_1"],
  "policy_ids": ["policy_1"]
}
```

Response:

```
{
  "status": "success",
  "applied": {
    "prompts": [
      {
        "id": "prompt_1",
        "success": true
      }
    ],
    "policies": [
      {
        "id": "policy_1",
        "success": true
      }
    ]
  }
}
```

POST /rollback

Rollback von Änderungen.

Request Body:

```json
{
  "prompt_ids": ["prompt_1"],
  "policy_ids": ["policy_1"]
}
```

Response:

```json
{
  "status": "success",
  "rolled_back": {
    "prompts": [
      {
        "id": "prompt_1",
        "success": true
      }
    ],
    "policies": [
      {
        "id": "policy_1",
        "success": true
      }
    ]
  }
}
```

Knowledge Base

GET /kb/stats

Gibt Wissensbasis-Statistiken zurück.

Response:

```
{
  "status": "success",
  "stats": {
    "total_sources": 15,
    "total_chunks": 1200,
    "total_summaries": 8,
    "sources_by_type": {
      "file": 10,
      "web": 3,
      "api": 2
    },
    "chunks_by_type": {
      "markdown": 800,
      "code": 200,
      "text": 200
    }
  }
}
```

POST /kb/ingest

Führt Wissensaufnahme durch.

Request Body:

```
{
  "docs_path": "docs",
  "results_path": "data/results"
}
```

Response:

```
{
  "status": "success",
  "ingest_results": {
    "timestamp": "2024-01-01T12:00:00Z",
    "docs_processed": {
      "processed": 5,
      "skipped": 2,
      "errors": 0,
        "sources": ["source_1", "source_2"],
        "chunks": ["chunk_1", "chunk_2"]
      },
      "results_processed": {
        "processed": 3,
        "skipped": 1,
        "errors": 0,
        "sources": ["source_3"],
        "chunks": ["chunk_3"]
      },
      "summaries_created": 2,
      "errors": []
    }
  }
```

Scheduler

GET /scheduler/status

Gibt Scheduler-Status zurück.

Response:

```
{
  "status": "success",
  "scheduler": {
    "running": true,
    "jobs_count": 4,
    "enabled_jobs": 4,
    "jobs": [
      {
        "job_id": "hourly_metrics_rollup",
        "name": "Hourly Metrics Rollup",
        "enabled": true,
        "last_run": "2024-01-01T11:00:00Z",
        "next_run": "2024-01-01T12:00:00Z",
        "last_status": "completed",
        "last_duration": 2.5,
        "last_error": null
      }
    ],
    "recent_results": [
      {
        "job_id": "hourly_metrics_rollup",
        "status": "completed",
          "start_time": "2024-01-01T11:00:00Z",
          "duration": 2.5,
          "error": null
        }
      ]
    }
  }
```

POST /scheduler/jobs/{id}/run

Führt Job manuell aus.

Response:

```
{
  "status": "success",
  "job_result": {
    "job_id": "hourly_metrics_rollup",
    "status": "completed",
    "start_time": "2024-01-01T12:00:00Z",
    "end_time": "2024-01-01T12:00:02Z",
    "duration": 2.0,
    "result_data": {
      "timestamp": "2024-01-01T12:00:00Z",
      "period": "hourly",
      "metrics": {
        "total_runs": 50,
        "avg_latency_ms": 1200,
        "total_cost": 0.5,
        "error_rate": 0.02,
        "win_rate": 0.8,
        "avg_quality_score": 0.85
      },
      "status": "completed"
    },
    "error": null
  }
}
```

Fehlerbehandlung

HTTP Status Codes

- `200 OK` - Erfolgreiche Anfrage
- `400 Bad Request` - Ungültige Anfrage
- `404 Not Found` - Ressource nicht gefunden
- `500 Internal Server Error` - Server-Fehler

Fehler-Response

```json
{
  "status": "error",
  "error": "Fehlerbeschreibung",
  "details": {
    "code": "ERROR_CODE",
    "message": "Detaillierte Fehlermeldung"
  }
}
```

Häufige Fehler

1. **Experiment nicht gefunden**
    - Status: 404
    - Lösung: Prüfe Experiment-ID
2. **Ungültiger Traffic Split**
    - Status: 400
    - Lösung: Summe muss 1.0 ergeben
3. **Guardrail-Verletzung**
    - Status: 400
    - Lösung: Anpassung der Limits
4. **Datenbank-Fehler**

- Status: 500
- Lösung: Prüfe Datenbank-Verbindung

Rate Limiting

Aktuell keine Rate Limits implementiert. In Produktionsumgebungen sollten angemessene Limits gesetzt werden.

Versionierung

API-Version: `v2`

Alle Breaking Changes werden über neue Versionen gehandhabt. Aktuelle Version ist stabil.

Beispiele

Vollständiger Workflow

```
# 1. Status prüfen
curl -X GET http://localhost:8000/v2/autopilot/status

# 2. Experiment erstellen
curl -X POST http://localhost:8000/v2/autopilot/experiments \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Prompt Test",
    "type": "prompt",
    "variants": {"A": "Original", "B": "Improved"},
    "traffic_split": {"A": 0.5, "B": 0.5}
  }'

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

# 4. Metriken loggen
curl -X POST http://localhost:8000/v2/autopilot/metrics \
  -H "Content-Type: application/json" \
  -d '{
    "task": "ai_ask",
    "provider": "openai",
    "latency_ms": 1500,
    "cost_est": 0.01,
    "quality_score": 0.85,
    "win": true,
    "experiment_id": "exp_123"
  }'

# 5. Experiment auswerten
curl -X POST http://localhost:8000/v2/autopilot/experiments/exp_123/eval

# 6. Änderungen anwenden
curl -X POST http://localhost:8000/v2/autopilot/apply \
  -H "Content-Type: application/json" \
  -d '{"prompt_ids": ["prompt_1"]}'
```

Rico V5 Autopilot API - Vollständige REST-API für selbstverbessernde Orchestrierung