Eine reproduzierbare MQM-Simulationsstudie

über eine fiktive virtuelle Fabrik mit konkreten plausiblen Parametern, Nachfrageprofil, Rückkopplungen, Ergebnisdaten, Diagrammen und ausführbarem Python-Code

Vorabgedanken

Die Quadranten des MQM wurden dazu als hierarchische Teilstrukturen mit Achsen- und Ebenen-Zuständen behandelt und ein deterministisches 30-Tage-Szenario aufgesetzt. Als reale Bezugsparameter werden von der KI „Kurt ChatGPT" quellenbasierte Leistungs-, CO_2- und Kostenwerte benutzt. Die Simulation folgt der MQM-Präzisierung: Die Quadranten sind Teilstrukturen des Systems, nicht selbst der Zustandsraum; der Systemzustand entsteht aus den Zuständen von (Q1(S), Q2(S), Q3(S), Q4(S)). Die innere Struktur jedes Quadranten nutzt (x, y, z, xy, xz, yz) als typisierte Zustandskomponenten für Zeit, Raum/Menge, Modus sowie Energie, Information und Form.

Als „Parameter" wurden reale technische und energiewirtschaftliche Eckwerte verwendet:

- eine VF-2-artige CNC mit 22,4 kW Spindelleistung und 0,6 kW Kühlmittelpumpe,
- ein UR10e-artigen Cobot mit 615 W durchschnittlicher Leistungsaufnahme,
- 363 g CO2/kWh für den deutschen Strommix 2024 und
- 0,241 EUR/kWh als deutscher Nicht-Haushalts-Strompreis Juni 2025.

Ergebnis der Simulation

Für die **fiktive Produktionszelle „VF-MQM-01" zur Fertigung eines Aluminium-Pumpengehäuses** wurden 40 Schichten à 8 Stunden simuliert.

Verglichen wurden am Ende zwei Varianten:

Kennzahl	MQM-Regelung	Statische Regelung
Nachfrage gesamt	2.613 Stück	2.613 Stück
Guter Output	2.612,2 Stück	2.524,3 Stück
Finaler Rückstand	0,8 Stück	88,7 Stück
Kumulativer Servicegrad	99,97 %	96,61 %
Ø Qualitätsausbeute	97,43 %	95,33 %
Energie gesamt	9.636,5 kWh	9.626,5 kWh
kWh / gute Einheit	3,69	3,81
CO2 gesamt	3.498 kg	3.494 kg

Interpretation

Die MQM-Regelung erzeugt etwas mehr Gesamtenergie, weil sie in Spitzenphasen aktiv in `rush` geht, **erreicht** aber praktisch: **vollständige Nachfrageerfüllung, bessere Qualität und einen niedrigeren spezifischen Energieverbrauch pro „guter" Einheit.**

Dateien hierzu im Anhang:

PDF-Bericht
CSV-Ergebnisdaten
Parameterdatei JSON
Python-Simulationscode
Komplettes Paket als ZIP

Konkrete MQM-Simulation einer fiktiven virtuellen Fabrik

Anwendungsfall: VF-MQM-01 - Produktionszelle fuer Aluminium-Pumpengehaeuse

Kurzfassung

Dieser Bericht beschreibt eine reproduzierbare diskrete Simulation einer fiktiven virtuellen Fabrik. Die Fabrik ist fiktiv, die technischen Eckwerte sind jedoch realistisch parametrisiert: zwei CNC-Bearbeitungszentren mit VF-2-aehnlicher Leistung, ein UR10e-aehnlicher Cobot, deutsche Strompreis- und CO2-Faktoren. Die Simulation bildet Q3 Planung, Q1 Steuerung, Q2 Prozess und Q4 Ergebnis als Teilstrukturen eines Systems ab. Zustaende sind typisiert: numerisch, ordinal, nominal und relational.

1. Reale Parameter und Quellen

Parameter	Wert in der Simulation	Begruendung / Quelle
CNC-Spindelleistung	22.4 kW nominal; 11.2 kW mittlere Last	VF-2: 30 hp / 22,4 kW, 8100 rpm; mittlere Last als Modellannahme
CNC-Kuehlmittelpumpe	0.6 kW je Maschine	VF-2-Spezifikation: 0,6 kW Pumpe
Cobot-Leistung	0.615 kW	UR10e: 615 W average, ca. 350 W typical program
Strompreis	0.241 EUR/kWh	Deutschland, Nicht-Haushalt, Jun 2025, Eurostat/CEIC
CO2-Faktor Strom	0.363 kg CO2/kWh	UBA: 363 g CO2/kWh fuer deutschen Strommix 2024
Schichtmodell	40 Schichten x 8.0 h	Modellannahme
Taktzeit automatisch	12.0 min/Stueck	Plausible Prozessannahme fuer kleines Fraesteil

Hinweis: Die reale Maximalleistung einer Spindel ist nicht gleich dem durchschnittlichen Prozessverbrauch. Darum wird die 22,4-kW-Angabe als technische Obergrenze verwendet und fuer den laufenden Prozess eine mittlere Last von 11,2 kW je CNC angesetzt.

2. MQM-Abbildung

Quadrant	Rolle in der Fabrik	Beispielzustand
Q3(S) Planung	Soll-Menge, Prioritaet, Sequenzregel, Portfolio	Q3_y = geplante Einheiten; Q3_z = Prioritaet
Q1(S) Steuerung	Moduswahl, Kapazitaet, Regelstrategie, Maschinenkonfiguration	Q1_z = automatic/rush/energy_save/stabilize/maintenance
Q2(S) Prozess	tatsaechliche Bearbeitung, Energie, Stabilitaet	Q2_xy = kWh; Q2_yz = stable/strained/unstable
Q4(S) Ergebnis	Guter Output, Qualitaet, Rueckstand, Kosten, CO2	Q4_y = gute Einheiten; Q4_z = high/medium/low

3. Rueckkopplungslogik

Die Simulation verwendet eine zeitdiskrete Hauptschleife: Q4(t) und Q2(t) melden Output, Qualitaet, Energie und Stabilitaet zurueck; daraus erzeugt Q3(t+1) einen aktualisierten Plan; Q1(t+1) loest Zielkonflikte; Q2(t+1) fuehrt den Prozess aus; Q4(t+1) bewertet Ergebnis und KPI. Die Konfliktregel lautet: Stabilitaet und Qualitaet vor Energie, Energie vor Liefertermin, Liefertermin vor Durchsatz.

Feedback	Regel
Rueckstand hoch	Q3_z eskaliert auf high/critical; Q1 kann rush waehlen
Qualitaet low	Q1_z wird stabilize
Energie > 110% Budget	Q1_z wird energy_save
Schicht 20	Praeventive Wartung: Q1_z wird maintenance

4. Simulationsergebnisse

Kennzahl	MQM-Regelung	Statische Regelung	Differenz MQM - statisch
Nachfrage gesamt [Stueck]	2.613	2.613	0
Guter Output [Stueck]	2.612,2	2.524,3	87,9
Finaler Rueckstand [Stueck]	0,8	88,7	-87,9
Kumulativer Servicegrad	100,0%	96,6%	3,4 pp
Qualitaetsausbeute avg.	97,43%	95,33%	2,10 pp
Energie gesamt [kWh]	9.636,5	9.626,5	10,0
kWh / gute Einheit	3,69	3,81	-0,12
Stromkosten [EUR]	2.322	2.320	2
CO2 [kg]	3.498	3.494	4
Systemscore avg.	85,1	84,3	0,8

Interpretation: Die MQM-Regelung greift aktiv in Q1 ein. Dadurch steigt in Spitzenphasen der Output, waehrend Qualitaets- und Energiegrenzen ueber Rueckkopplungen begrenzt werden. Der statische Referenzfall bleibt in automatic und kann Demand-Spikes schlechter kompensieren.

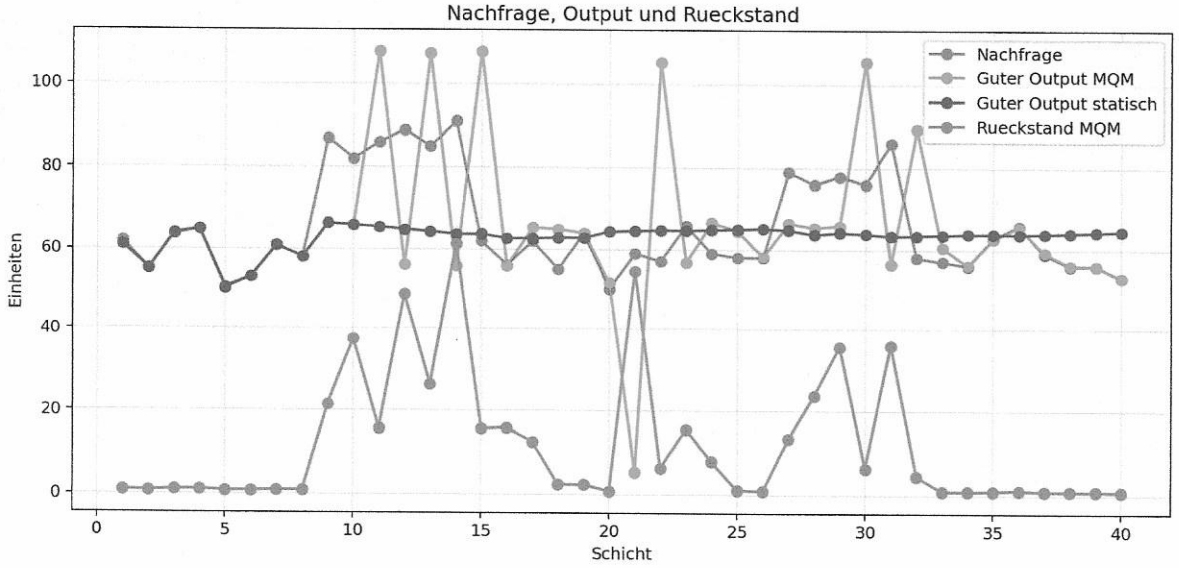


Abbildung 1: Simulationsergebnis, Quelle: eigene Simulation.

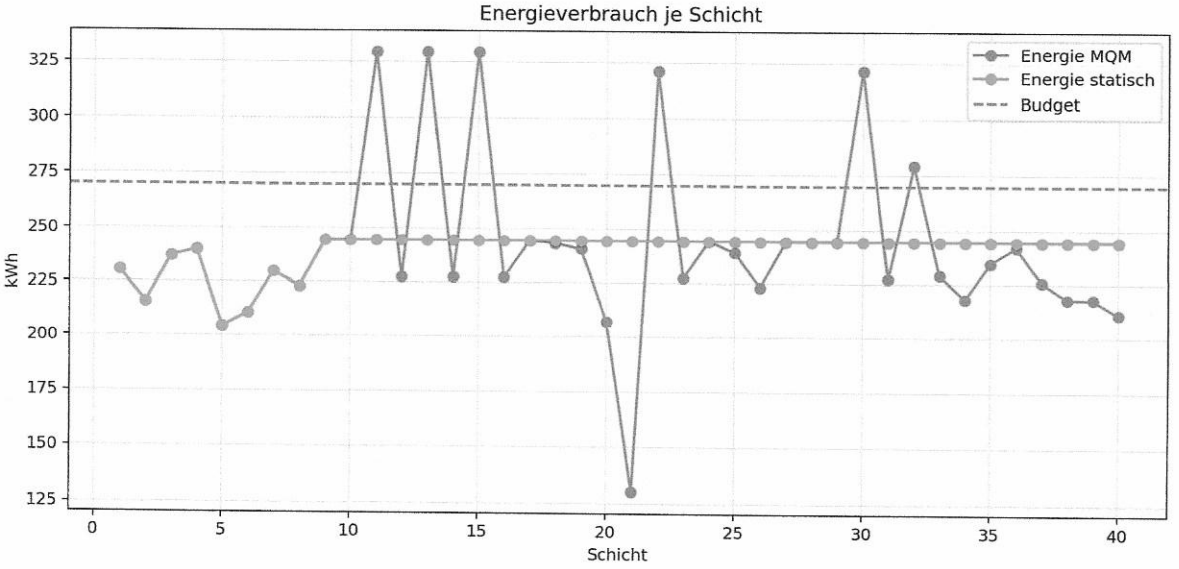


Abbildung 2: Simulationsergebnis, Quelle: eigene Simulation.

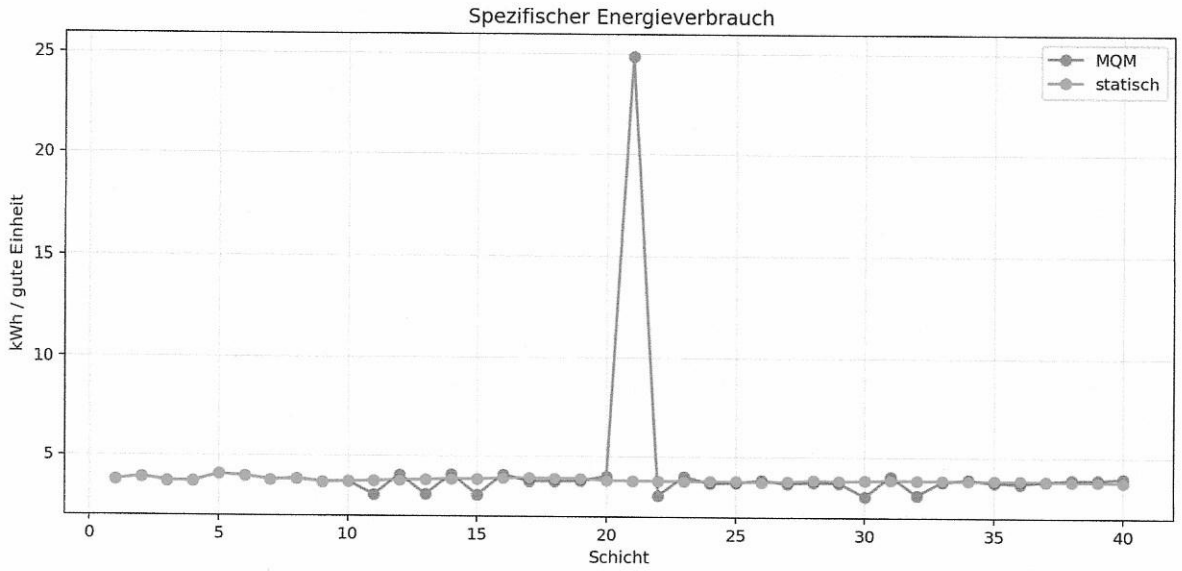


Abbildung 3: Simulationsergebnis, Quelle: eigene Simulation.

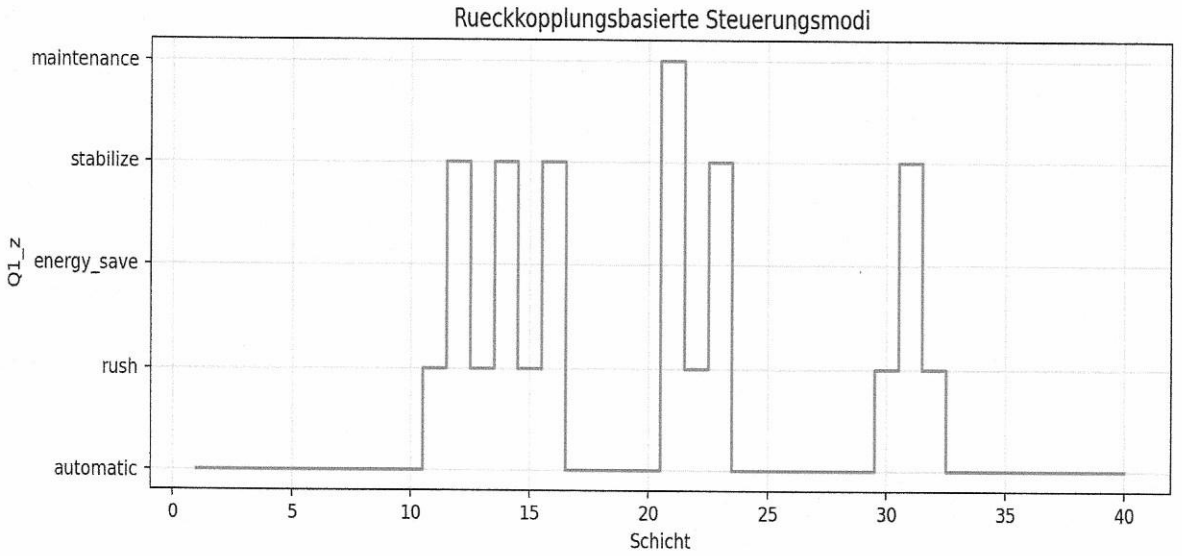


Abbildung 4: Simulationsergebnis, Quelle: eigene Simulation.

5. Auszug aus dem MQM-Zustandsraum

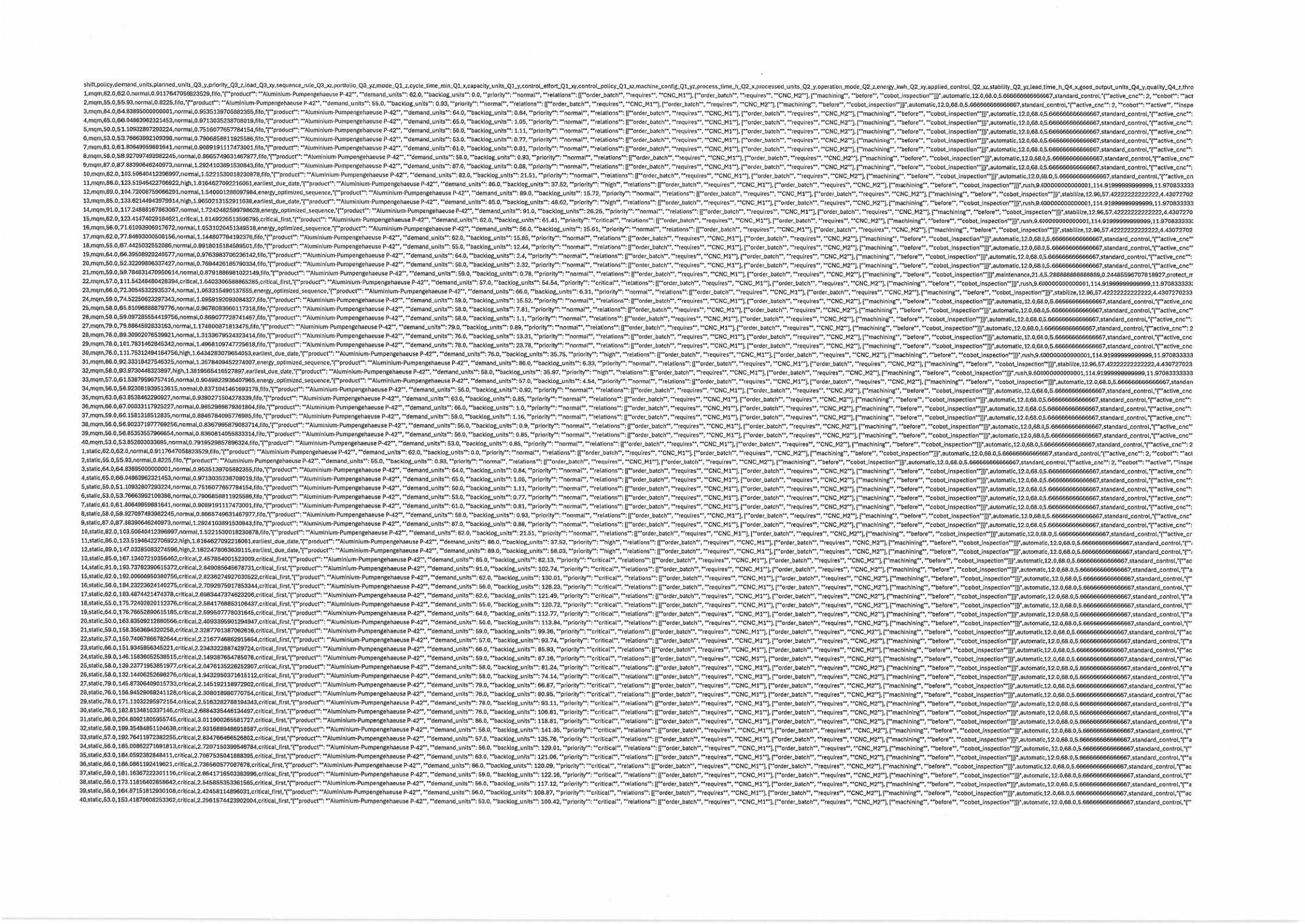
Schicht	Q3_y Plan	Q3_z	Q3_xy	Q1_z	Q2_y	Q2_xy kWh	Q2_yz	Q4_y	Q4_z	Backlog
1	62,0	normal	0,91	automatic	62,0	230,4	stable	61,1	high	0,9
2	55,9	normal	0,82	automatic	55,9	215,8	stable	55,1	high	0,8
3	64,8	normal	0,95	automatic	64,8	237,2	stable	63,8	high	1,0
4	66,0	normal	0,97	automatic	66,0	240,2	stable	64,9	high	1,1
5	51,1	normal	0,75	automatic	51,1	204,3	stable	50,3	high	0,8
6	53,8	normal	0,79	automatic	53,8	210,6	stable	53,0	high	0,8
7	61,8	normal	0,91	automatic	61,8	230,0	stable	60,9	high	0,9
8	58,9	normal	0,87	automatic	58,9	223,0	stable	58,0	high	0,9
9	87,9	normal	1,29	automatic	68,0	244,8	strained	66,4	high	21,5
10	103,5	normal	1,52	automatic	68,0	244,8	strained	66,0	medium	37,5
11	123,5	high	1,82	rush	114,9	329,5	unstable	107,8	low	15,7
12	104,7	normal	1,54	stabilize	57,4	227,5	strained	56,1	high	48,6

6. Reproduzierbarkeit

Die Simulation ist deterministisch reproduzierbar. Alle Eingangsparameter stehen in der JSON-Datei; alle Schichtergebnisse stehen in der CSV-Datei; die Python-Datei enthaelt die vollstaendige Simulationslogik. Die wichtigste methodische Einschraenkung ist, dass keine realen Betriebsdaten einer konkreten Fabrik kalibriert wurden. Das Ergebnis ist daher ein plausibler digitaler Demonstrator, kein validierter industrieller Digital Twin.

7. Quellenhinweise

- Haas VF-2 Spezifikation: 30 hp / 22,4 kW Spindel, 0,6 kW Kuehlmittelpumpe.
- Universal Robots UR10e technische Spezifikation: 615 W average power consumption, ca. 350 W typical program.
- Umweltbundesamt: 363 g CO2/kWh deutscher Strommix 2024.
- Eurostat/CEIC: Deutschland Nicht-Haushalt Strompreis Jun 2025; in der Simulation 0,241 EUR/kWh.
- MQM-Modellgrundlage: Quadranten als Teilstrukturen; x, y, z sowie xy, xz, yz als typisierte Zustandskomponenten.



3529,3.773137864938017,55.53255258823529,83.64446717647058,0.985,89.28622448901444,0.0
35,0.8225,3.9179302264160873,52.018136489999996,78.35096906999999,0.992829487179487,89.50423546265527,0.0
1.0486396232145268}",1.0486396232145268,18.951360376785473,0.01617298897058824,0.9838270110294117,0.9535139705882355,3.719151935299484,57.17625136617353,86.12024583369707,0.9942064109214668,89.870015883113
.109328072932243}",1.109328072932243,18.890671927067757,0.016795623335478065,0.983204376664522,0.9713035238708019,3.6980993968142206,57.876638958544746,87.17518648112757,0.9954905362888934,90.0047423132423
329}",0.7666399210939829,19.233360078906017,0.015,0.985,0.7516077657784154,4.0574243956252944,49.22705790271863,74.14697933065088,0.9974100002665746,89.16200904742666,0.0
35}",0.8064995988164085,19.1935004011836,0.015,0.985,0.7906858811925586,3.977436747963704,50.76559159912901,76.46435581113623,0.9976891128973744,89.32522714411378,0.0
52}",0.9270974939822452,19.072902506017755,0.015,0.985,0.9089191117473001,3.7773147209214426,55.420519486928114,83.4757202230494,0.9977387866000436,89.79571437488471,0.0
97333}",0.8839064624097333,19.116093537590274,0.015,0.985,0.8665749631467977,3.8427100287816605,53.75339969936274,80.96466427746337,0.9981113109777572,89.69908741692514,0.0
0.023860259728827105,0.9761397402711729,1.0,3.688599169514264,59.00643999999999,88.87691999999998,0.9612497222991533,77.87570364023486,0.0
0.0,0.02960382504557719,0.9703961749544228,1.0,3.710431191119677,59.00643999999999,88.87691999999998,0.9410997421867046,79.31154417793722,0.0
ry"": 21.799376636406308, ""backlog"": 15.720087590662914}",15.720087590662914,21.799376636406308,0.06196156773054015,0.9380384322694598,1.0,3.0568380400885524,79.41558168319999,119.61766037759998,0.978257140261
95}","{""inventory"": 0.0, ""backlog"": 48.621449439799136}",48.621449439799136,0.0,0.0230500322024496,0.9769499677975504,1.0,4.055036904607947,54.82317356799999,82.57598342399999,0.9401213676849765,71.6390
ntory"": 22.37256776116847, ""backlog"": 26.248881678630667}",26.248881678630667,22.37256776116847,0.06567553288227909,0.9343244671177209,1.0,3.068989054382785,79.41558168319999,119.61766037759998,0.9707370326882
354377578}","{""inventory"": 0.0, ""backlog"": 61.41474029184621}",61.41474029184621,0.0,0.027656206499496565,0.9723437935005035,1.0,4.0742463723789495,54.82317356799999,82.57598342399999,0.9378393316884148,
45.80380122266949, ""backlog"": 15.610939069176723}",15.610939069176723,45.80380122266949,0.06192306628376699,0.938076933716233,1.0,3.0567125784309153,79.41558168319999,119.61766037759998,0.985132438981
581270616}","{""inventory"": 45.567810286764654, ""backlog"": 15.846930005081568}",15.846930005081568,45.567810286764654,0.028877551128373792,0.9711224488716262,1.0,4.079370400692175,54.82317356799999,82.575
3255208607,48.97223703663761,0.0381701948548094,0.9618298051451906,1.0,3.7434775009395764,59.00643999999999,88.87691999999998,0.9893471718705409,86.773014736144,0.018
"": 2.395089202405771}",2.395089202405771,59.01965108944044,0.035513053146063255,0.9644869468539368,0.9918015184589501,3.743433414030325,58.68365984602821,88.39074076393462,0.9980416277985238,89.352851682987
2.32209806337427}",2.32209806337427,59.092642228471945,0.034973942971826495,0.9650260570281735,0.9763983706236142,3.7610908923487907,58.07722676387203,87.47731666093587,0.9981957280004865,89.32011367230828,
5141}",0.7848314709506141,60.629908820895594,0.015,0.985,0.7694426185790334,4.019910981697299,49.929228969912394,75.2046062907809,0.9994129906724382,89.25170737052821,0.0
5487562256, ""backlog"": 54.542484804283944}",54.542484804283944,6.872255487562256,0.008799999999999999,0.9912,1.0,24.85335829246953,31.39987072,47.295240959999994,0.9609294521459287,42.3593377660323 4,0.0
inventory"": 55.10928696249246, ""backlog"": 6.305453329353739}",6.305453329353739,55.10928696249246,0.05652961147868896,0.943470388521311,0.9706098573293069,3.0508217742513137,77.37532192659039,116.544572
125}","{""inventory"": 45.89223405887276, ""backlog"": 15.522506232973434}",15.522506232973434,45.89223405887276,0.011132887253438872,0.9888671127465611,1.0,4.00616839442807,54.82317356799999,82.57598342399
88879776,53.60377140304843,0.018947980232710812,0.9810520197672892,1.0,3.670129781852135,59.00643999999999,88.87691999999998,0.9950500830869472,87.55788781252059,0.0
72855544197557}",1.0972855544197557,60.31745473742643,0.016673292810410613,0.9833267071895894,0.9678083660117318,3.7021698280905504,57.73903206313887,86.96791966356601,0.9993292875584233,90.03416398426842,
3}",0.8864592833162988,60.52828100852989,0.015,0.985,0.869077728741467,3.8386676223670997,53.85193539282716,81.11308111035792,0.9994767064443236,89.7097751711441,0.0
0765399211,48.105719526446975,0.020920021795336873,0.9790799782046631,1.0,3.677522077304153,59.00643999999999,88.87691999999998,0.992493502106374,81.18746172440063,0.0
3146284534197,37.63159400731199,0.03638419881080853,0.9636158011891914,1.0,3.7365392211819866,59.00643999999999,88.87691999999998,0.9871372924367042,81.33324210532184,0.0
753124941647556,25.66161535019863,0.028970274369314042,0.9710297256306859,1.0,3.708010311379013,59.00643999999999,88.87691999999998,0.9814462247318904,81.01021127997521,0.0
}","{""inventory"": 55.08155601638293, ""backlog"": 6.333184275463253}",6.333184275463253,55.08155601638293,0.05667120520137726,0.9433287947986227,0.9724427857783464,3.0505367266096948,77.50256357332964,116.73622646
676}","{""inventory"": 25.441695459456483, ""backlog"": 35.973044832389704}",35.973044832389704,25.441695459456483,0.01849602363068502,0.981503976369315,1.0,4.036222235215861,54.82317356799999,82.5759834239
}","{""inventory"": 56.87594432427203, ""backlog"": 4.53879596757416}",4.53879596757416,56.87594432427203,0.048298913541319745,0.9517010864586802,0.8177257642915917,3.1244413255703467,67.34312720097199,101.4338388960
.491658352332564, ""backlog"": 0.9230819395136152}",0.9230819395136152,60.491658352332564,0.015,0.985,0.9049822936407965,3.7831366188671502,55.26552410781875,83.24226245285563,0.9995811787933242,89.858906055
927029}",0.8538462290927029,60.560894062753476,0.015,0.985,0.8371041461693178,3.892128704266338,52.59311229808787,79.217011469734,0.9996221919340298,89.5702236265338,0.0
0331179252278}",1.000331179252278,60.4144091125939,0.015665950264974186,0.9843340497350258,0.9390271504278339,3.7369307224304387,56.60589476800072,85.26116099910482,0.9995693796042824,89.87338739993812,0.0
581318512834997}",1.1581318512834997,60.25660844056267,0.017285464577556316,0.9827145354224437,0.9852989879301804,3.6821124801224836,58.427650511278465,88.00513334271405,0.9995152231681528,89.9775452556553
",0.9023719777692563,60.51236831407692,0.015,0.985,0.8846784095776985,3.813985411454784,54.46614549224252,82.03821914391715,0.9996313839960095,89.77552416767742,0.0
55414}",0.8535355796665414,60.56120471217963,0.015,0.985,0.8367995879083714,3.8926575804247854,52.58112161885422,79.19895082009992,0.9996591311582803,89.56886222824258,0.0
39968}",0.8528030336949968,60.561937258151175,0.015,0.985,0.8360814055833314,3.893906254935263,52.5528462607534,79.15636179524267,0.999666873814963,89.56564943957501,0.0
",0.8077920455054226,60.60694824634075,0.015,0.985,0.7919529857896324,3.9749752831134977,50.815478424499524,76.53949654810509,0.9996908564693818,89.36138176271936,0.0
3529,3.773137864938017,55.53255258823529,83.64446717647058,0.985,89.28622448901444,0.0
85,0.8225,3.9179302264160873,52.018136489999996,78.35096906999999,0.992829487179487,89.50423546265527,0.0
1.0486396232145268}",1.0486396232145268,18.951360376785473,0.01617298897058824,0.9838270110294117,0.9535139705882355,3.719151935299484,57.17625136617353,86.12024583369707,0.9942064109214668,89.870015883
.109328072932243}",1.109328072932243,18.890671927067757,0.016795623335478065,0.983204376664522,0.9713035238708019,3.6980993968142206,57.876638958544746,87.17518648112757,0.9954905362888934,90.0047423132
329}",0.7666399210939829,19.233360078906017,0.015,0.985,0.7516077657784154,4.0574243956252944,49.22705790271863,74.14697933065088,0.9974100002665746,89.16200904742666,0.0
35}",0.8064995988164085,19.1935004011836,0.015,0.985,0.7906858811925586,3.977436747963704,50.76559159912901,76.46435581113623,0.9976891128973744,89.32522714411378,0.0
52}",0.9270974939822452,19.072902506017755,0.015,0.985,0.9089191117473001,3.7773147209214426,55.420519486928114,83.4757202230494,0.9977387866000436,89.79571437488471,0.0
97333}",0.8839064624097333,19.116093537590274,0.015,0.985,0.8665749631467977,3.8427100287816605,53.75339969936274,80.96466427746337,0.9981113109777572,89.69908741692514,0.0
0.023860259728827105,0.9761397402711729,1.0,3.688599169514264,59.00643999999999,88.87691999999998,0.9612497222991533,77.87570364023486,0.0
0.0,0.02960382504557719,0.9703961749544228,1.0,3.710431191119677,59.00643999999999,88.87691999999998,0.9410997421867046,79.31154417793722,0.0
.58.03285083274595,0.0,0.03696156773054015,0.9630384322694598,1.0,3.7387793826764604,59.00643999999999,88.87691999999998,0.9197332630252477,77.28125482212822,0.0
52}",82.13407210356462,0.0,0.045606195159097786,0.9543938048409022,1.0,3.7726441821301817,59.00643999999999,88.87691999999998,0.898849664897088,75.63738165783786,0.0
3782390615374,0.0,0.05299635003807522,0.9470036499619248,1.0,3.8020848551522293,59.00643999999999,88.87691999999998,0.8854650792573537,76.47175643854371,0.0
66950380756,0.0,0.06277714114196828,0.9372228588580317,1.0,3.8417631423131198,59.00643999999999,88.87691999999998,0.8684143021216523,73.781968889806,0.0
624140275}",128.23223624140275,1.7744332624048127,0.062140687317576306,0.9378593126824237,1.0,3.8391560296990326,59.00643999999999,88.87691999999998,0.8778740607224736,85.7942620517428 8,0.0
1474378}",121.4874421474378,8.519227356369754,0.07728243979463337,0.9227175602053667,1.0,3.9021564025428805,59.00643999999999,88.87691999999998,0.8901560197581937,85.17600062442855,0.018
376}",120.72402820112376,9.2826413026838,0.07700861843655801,0.922991381563442,1.0,3.900998760351513,59.00643999999999,88.87691999999998,0.8966403868141064,85.18718117859028,0.018
5185}",112.76652890615185,17.24014059765571,0.0741544221327661,0.9258455778672339,1.0,3.888972763242426,59.00643999999999,88.87691999999998,0.9077951521617729,85.30372247612445,0.018
0566}",113.83509212880566,16.171577375001903,0.07453769445079111,0.9254623055492088,1.0,3.8905833481325582,59.00643999999999,88.87691999999998,0.911550044965963,84.62022085398608,0.018
5943202579}",99.35636943202579,30.65030007178177,0.05178348975323736,0.9482165102467627,1.0,3.7972216222612536,59.00643999999999,88.87691999999998,0.9256870834465027,86.21716262146273,0.0
5782643}",93.74067866782643,36.26599083598113,0.04976925346765654,0.9502307465323434,1.0,3.7891725230252398,59.00643999999999,88.87691999999998,0.9328505167135914,86.29940703820105,0.0
9563452211}",85.93459563452211,44.07207386928546,0.04696936715728913,0.9530306328427108,1.0,3.7780404020742138,59.00643999999999,88.87691999999998,0.9408571261978512,86.41373077379167,0.0
2538516}",87.15836052538516,42.84830897842241,0.04740830721857431,0.9525916927814257,1.0,3.7797812668100605,59.00643999999999,88.87691999999998,0.9426212241439202,85.6541324733876,0.0
53851977}",81.23771953851977,48.76894996528779,0.045284691369626894,0.9547153086303731,1.0,3.771373730729731,59.00643999999999,88.87691999999998,0.9485185554255261,86.48251872014201,0.0
252698276}",74.14406252698276,55.86260697682479,0.042740338065632266,0.9572596619343677,1.0,3.7613495882802415,59.00643999999999,88.87691999999998,0.9546796683820399,86.58640864356421,0.0
409015733}",66.87306409015733,63.13360541365023,0.040132375929037775,0.9598676240709623,1.0,3.7511299943875684,59.00643999999999,88.87691999999998,0.9605235749172626,86.69289581507198,0.0
069241126}",80.94529069241126,49.06137881139631,0.045179802974322544,0.9548201970256774,1.0,3.7709594398088426,59.00643999999999,88.87691999999998,0.9543455777256564,79.36162345085445,0.0
156}",93.11032295972156,36.89634654408603,0.06125047451926885,0.9387495254807312,1.0,3.8355153718456108,59.00643999999999,88.87691999999998,0.9496428756302209,79.42796227037654,0.0
3103371458}",106.81348103371458,23.19318847009299,0.05445820697048585,0.9455417930295141,1.0,3.8079630766587687,59.00643999999999,88.87691999999998,0.9445700669259395,79.08068918487663,0.0
5955745}",118.80921805955745,11.197451444250135,0.05876083861533624,0.9412391613846638,1.0,3.8253701960267628,59.00643999999999,88.87691999999998,0.9406843644235859,79.6187207554723,0.0
84851104638,0.0,0.06684750663954317,0.9331524933604568,1.0,3.8585207250829123,59.00643999999999,88.87691999999998,0.9323337249827447,75.1157355149775,0.0
2382255}",135.76411972382255,5.590728787223846,0.06484222371729646,0.9351577762827036,1.0,3.850246799643399,59.00643999999999,88.87691999999998,0.9367656638454483,85.68395409677115,0.0
71691813}",129.00862271691813,12.346225794128287,0.062419161663170054,0.9375808383368299,1.0,3.84029631160251,59.00643999999999,88.87691999999998,0.9414661421429591,85.78289150882601,0.0
828484109}",121.05923828484109,20.295610226205326,0.05956787599886696,0.9404321240011331,1.0,3.828652960061772,59.00643999999999,88.87691999999998,0.9464339653606897,85.89931396039916,0.0
196211}",120.08611924196211,21.268729269084304,0.059218837604720985,0.940781162395279,1.0,3.8272324948841745,59.00643999999999,88.87691999999998,0.9483055879285568,85.9135657437432,0.0
2301116}",122.16367222301116,19.19117628803525,0.059964014427191945,0.9400359855728081,1.0,3.8302663840045548,59.00643999999999,88.87691999999998,0.948864097018413,84.62401599020248,0.0
02858643}",117.11816402858643,24.236684482459978,0.058154291258459986,0.94184570874154,1.0,3.822906662817514,59.00643999999999,88.87691999999998,0.9521576127334205,85.95703283544734,0.0
2930108}",108.87151812930108,32.48333038174532,0.05519638383403912,0.9448036161659609,1.0,3.8109382454582503,59.00643999999999,88.87691999999998,0.9565209592135377,86.07780882018601,0.0
7060825336}",100.4187060825336,40.936142428512795,0.05216452872400775,0.9478354712759922,1.0,3.798748141855193,59.00643999999999,88.87691999999998,0.9607739429365102,86.20160420454317,0.0
57373459696}",88.67957373459696,52.675274776449456,0.04795393605975501,0.952046063940245,1.0,3.781947504086428,59.00643999999999,88.87691999999998,0.9660621608363577,86.37352928616002,0.0

```
{
  "scenario_name": "VF-MQM-01 - fiktive virtuelle Fabrik",
  "product_name": "Aluminium-Pumpengehaeuse P-42",
  "n_shifts": 40,
  "shift_hours": 8.0,
  "cnc_count": 2,
  "cycle_time_min_auto": 12.0,
  "oee_auto": 0.85,
  "cnc_spindle_nominal_kw": 22.4,
  "cnc_run_kw": 11.2,
  "cnc_idle_kw": 2.5,
  "coolant_pump_kw_per_machine": 0.6,
  "cobot_average_kw": 0.615,
  "compressed_air_kw_at_full_utilization": 4.0,
  "base_load_kw": 5.0,
  "energy_budget_kwh_per_shift": 270.0,
  "electricity_price_eur_per_kwh": 0.241,
  "co2_kg_per_kwh": 0.363,
  "base_defect_rate": 0.015,
  "initial_backlog_units": 0.0,
  "initial_inventory_units": 20.0,
  "random_seed": 42
}
```

```
from __future__ import annotations

import json
import math
import os
from dataclasses import dataclass, asdict
from typing import Dict, Any, List, Tuple

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from reportlab.lib import colors
from reportlab.lib.enums import TA_CENTER, TA_LEFT
from reportlab.lib.pagesizes import A4, landscape
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import cm
from reportlab.platypus import (
    SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle, PageBreak,
    Image, KeepTogether
)
from reportlab.pdfbase.ttfonts import TTFont
from reportlab.pdfbase import pdfmetrics


@dataclass(frozen=True)
class FactoryParams:
    scenario_name: str = "VF-MQM-01 - fiktive virtuelle Fabrik"
    product_name: str = "Aluminium-Pumpengehaeuse P-42"
    n_shifts: int = 40
    shift_hours: float = 8.0
    cnc_count: int = 2
    cycle_time_min_auto: float = 12.0
    oee_auto: float = 0.85
    cnc_spindle_nominal_kw: float = 22.4
    cnc_run_kw: float = 11.2
    cnc_idle_kw: float = 2.5
    coolant_pump_kw_per_machine: float = 0.6
    cobot_average_kw: float = 0.615
    compressed_air_kw_at_full_utilization: float = 4.0
    base_load_kw: float = 5.0
    energy_budget_kwh_per_shift: float = 270.0
    electricity_price_eur_per_kwh: float = 0.241
    co2_kg_per_kwh: float = 0.363
    base_defect_rate: float = 0.015
    initial_backlog_units: float = 0.0
    initial_inventory_units: float = 20.0
    random_seed: int = 42


MODE = {
    "automatic": {
        "capacity_factor": 1.00,
        "cycle_factor": 1.00,
```

```
        "oee_factor": 1.00,
        "run_energy_factor": 1.00,
        "defect_rate": 0.015,
        "policy": "standard_control",
    },
    "rush": {
        "capacity_factor": 1.30,
        "cycle_factor": 0.80,
        "oee_factor": 1.04,
        "run_energy_factor": 1.18,
        "defect_rate": 0.040,
        "policy": "maximize_throughput",
    },
    "energy_save": {
        "capacity_factor": 0.95,
        "cycle_factor": 1.05,
        "oee_factor": 1.00,
        "run_energy_factor": 0.84,
        "defect_rate": 0.020,
        "policy": "minimize_energy",
    },
    "stabilize": {
        "capacity_factor": 0.95,
        "cycle_factor": 1.08,
        "oee_factor": 0.96,
        "run_energy_factor": 0.95,
        "defect_rate": 0.008,
        "policy": "maximize_stability",
    },
    "maintenance": {
        "capacity_factor": 0.20,
        "cycle_factor": 1.80,
        "oee_factor": 0.70,
        "run_energy_factor": 0.40,
        "defect_rate": 0.006,
        "policy": "protect_machine",
    },
}

PRIORITY_LEVELS = ["low", "normal", "high", "critical"]
STABILITY_LEVELS = ["stable", "strained", "unstable", "critical"]
QUALITY_LEVELS = ["low", "medium", "high"]


def demand_profile(params: FactoryParams) -> np.ndarray:
    rng = np.random.default_rng(params.random_seed)
    base = rng.normal(60, 5, size=params.n_shifts)

    # Demand shock: market campaign in shifts 9-14 and late replenishment in
27-31.
    base[8:14] += rng.normal(24, 4, size=6)
    base[26:31] += rng.normal(14, 3, size=5)
    # Weekend-like lower demand every 10th shift.
    for idx in [19, 39]:
```

```
        base[idx] -= 10
    return np.clip(np.round(base), 42, 92)


def escalate_priority(backlog: float, demand: float, prev_priority: str) -> str:
    if backlog > 0.75 * max(demand, 1):
        return "critical"
    if backlog > 0.35 * max(demand, 1):
        return "high"
    if backlog > 0.05 * max(demand, 1):
        return "normal"
    return "normal"


def select_sequence_rule(priority: str, energy_over_budget: bool) -> str:
    if energy_over_budget:
        return "energy_optimized_sequence"
    if priority == "critical":
        return "critical_first"
    if priority == "high":
        return "earliest_due_date"
    return "fifo"


def choose_mode(policy: str, shift: int, load: float, priority: str,
prev_quality: str,
                prev_stability: str, prev_energy: float, params: FactoryParams)
-> str:
    if policy == "static":
        return "automatic"

    # Preventive maintenance only when the load allows it.
    if shift in (20,) and load < 0.90:
        return "maintenance"

    # Conflict resolution: protect quality and stability first; in critical
backlog phases, delivery overrides energy.
    if prev_stability == "critical":
        return "maintenance"
    if prev_quality == "low" or prev_stability == "unstable":
        return "stabilize"
    if load > 1.02 and priority in ("high", "critical"):
        return "rush"
    if prev_energy > 1.12 * params.energy_budget_kwh_per_shift and load < 0.98:
        return "energy_save"
    if load > 0.95 and priority == "critical":
        return "rush"
    return "automatic"


def nominal_capacity_units(params: FactoryParams, mode: str) -> Tuple[float,
float, float]:
    mp = MODE[mode]
    cycle_time = params.cycle_time_min_auto * mp["cycle_factor"]
```

```
    oee = params.oee_auto * mp["oee_factor"]
    capacity = params.cnc_count * params.shift_hours * 60.0 / cycle_time * oee *
mp["capacity_factor"]
    return capacity, cycle_time, oee


def simulate(policy: str, params: FactoryParams) -> pd.DataFrame:
    demands = demand_profile(params)
    records: List[Dict[str, Any]] = []

    backlog = params.initial_backlog_units
    inventory = params.initial_inventory_units
    prev_quality = "high"
    prev_stability = "stable"
    prev_energy = params.energy_budget_kwh_per_shift * 0.92
    prev_priority = "normal"
    cumulative_good = 0.0
    cumulative_demand = 0.0

    for t in range(params.n_shifts):
        demand = float(demands[t])
        cumulative_demand += demand

        # Q3: planning
        planned_units = demand + backlog
        # load computed against nominal automatic capacity, because Q3 estimates
before Q1 resolves mode.
        auto_capacity, _, _ = nominal_capacity_units(params, "automatic")
        priority = escalate_priority(backlog, demand, prev_priority)
        q3_load = planned_units / max(auto_capacity, 1.0)
        energy_over_budget = prev_energy > params.energy_budget_kwh_per_shift
        q3_sequence_rule = select_sequence_rule(priority, energy_over_budget)
        q3_portfolio = {
            "product": params.product_name,
            "demand_units": demand,
            "backlog_units": round(backlog, 2),
            "priority": priority,
            "relations": [
                ["order_batch", "requires", "CNC_M1"],
                ["order_batch", "requires", "CNC_M2"],
                ["machining", "before", "cobot_inspection"],
            ],
        }

        # Q1: control / transformation
        mode = choose_mode(policy, t, q3_load, priority, prev_quality,
prev_stability,
                           prev_energy, params)
        capacity_units, cycle_time_min, oee = nominal_capacity_units(params,
mode)
        q1_control_effort = capacity_units / max(cycle_time_min, 0.01)
        q1_machine_config = {
            "active_cnc": params.cnc_count if mode != "maintenance" else 1,
            "cobot": "active" if mode != "maintenance" else "limited",
```

```
            "inspection": "increased" if mode == "stabilize" else "normal",
            "mode": mode,
        }
        q1_policy = MODE[mode]["policy"]

        # Q2: process execution
        # deterministic disturbance: raw material surface issue around shifts
16-18
        raw_material_defect_extra = 0.018 if 15 <= t <= 18 else 0.0
        # deterministic disturbance: tool wear in shift 27 if not stabilized
        tool_wear_extra = 0.012 if t == 27 and mode != "stabilize" else 0.0

        processed_units = min(planned_units, capacity_units)
        utilization = processed_units / max(capacity_units, 1.0)
        total_machine_run_hours = processed_units * cycle_time_min / 60.0
        max_machine_hours = params.cnc_count * params.shift_hours
        idle_machine_hours = max(0.0, max_machine_hours -
total_machine_run_hours)

        run_energy = total_machine_run_hours * params.cnc_run_kw *
MODE[mode]["run_energy_factor"]
        idle_energy = idle_machine_hours * params.cnc_idle_kw
        coolant_energy = params.cnc_count * params.coolant_pump_kw_per_machine *
params.shift_hours * max(utilization, 0.25)
        cobot_energy = params.cobot_average_kw * params.shift_hours * (0.3 + 0.7
* utilization)
        compressed_air_energy = params.compressed_air_kw_at_full_utilization *
params.shift_hours * utilization
        base_energy = params.base_load_kw * params.shift_hours
        energy_kwh = run_energy + idle_energy + coolant_energy + cobot_energy +
compressed_air_energy + base_energy

        overload_defect = max(0.0, utilization - 0.92) * 0.035 + max(0.0,
q3_load - 1.05) * 0.025
        defect_rate = min(0.18, MODE[mode]["defect_rate"] +
raw_material_defect_extra + tool_wear_extra + overload_defect)
        quality_yield = max(0.0, 1.0 - defect_rate)
        good_output = processed_units * quality_yield
        scrap_rework = processed_units - good_output

        if mode == "maintenance":
            stability = "strained"
        elif utilization > 0.98 and mode == "rush":
            stability = "unstable"
        elif utilization > 0.97 and q3_load > 1.05:
            stability = "strained"
        elif mode == "stabilize":
            stability = "stable"
        else:
            stability = "stable"

        if defect_rate > 0.055 or stability == "unstable":
            quality = "low"
        elif defect_rate > 0.025:
```

```
            quality = "medium"
        else:
            quality = "high"

        process_time_h = total_machine_run_hours / max(params.cnc_count, 1)
        throughput_good_units_h = good_output / max(params.shift_hours, 1.0)
        energy_per_good = energy_kwh / max(good_output, 0.01)
        cost_eur = energy_kwh * params.electricity_price_eur_per_kwh
        co2_kg = energy_kwh * params.co2_kg_per_kwh

        # Q4: result and feedback variables
        backlog = max(0.0, planned_units - good_output)
        inventory = max(0.0, inventory + good_output - demand)
        cumulative_good += good_output
        service_level_cum = cumulative_good / max(cumulative_demand, 1.0)
        efficiency_index = min(1.0, 1.95 / max(energy_per_good, 0.01))
        stability_score = {"stable": 1.0, "strained": 0.75, "unstable": 0.35,
"critical": 0.1}[stability]
        service_shift = min(1.0, good_output / max(demand, 1.0))
        system_score = 100.0 * (0.40 * service_shift + 0.30 * quality_yield +
0.20 * efficiency_index + 0.10 * stability_score)

        rec = {
            "shift": t + 1,
            "policy": policy,
            "demand_units": demand,
            "planned_units_Q3_y": planned_units,
            "priority_Q3_z": priority,
            "load_Q3_xy": q3_load,
            "sequence_rule_Q3_xz": q3_sequence_rule,
            "portfolio_Q3_yz": json.dumps(q3_portfolio, ensure_ascii=False),
            "mode_Q1_z": mode,
            "cycle_time_min_Q1_x": cycle_time_min,
            "capacity_units_Q1_y": capacity_units,
            "control_effort_Q1_xy": q1_control_effort,
            "control_policy_Q1_xz": q1_policy,
            "machine_config_Q1_yz": json.dumps(q1_machine_config,
ensure_ascii=False),
            "process_time_h_Q2_x": process_time_h,
            "processed_units_Q2_y": processed_units,
            "operation_mode_Q2_z": mode,
            "energy_kwh_Q2_xy": energy_kwh,
            "applied_control_Q2_xz": json.dumps({"policy": q1_policy,
"sequence": q3_sequence_rule}, ensure_ascii=False),
            "stability_Q2_yz": stability,
            "lead_time_h_Q4_x": params.shift_hours + backlog /
max(auto_capacity, 1.0) * params.shift_hours,
            "good_output_units_Q4_y": good_output,
            "quality_Q4_z": quality,
            "throughput_good_units_h_Q4_xy": throughput_good_units_h,
            "trace_Q4_xz": json.dumps({"defect_rate": defect_rate,
"scrap_rework": scrap_rework}, ensure_ascii=False),
            "inventory_backlog_Q4_yz": json.dumps({"inventory": inventory,
"backlog": backlog}, ensure_ascii=False),
```

```
            "backlog_units": backlog,
            "inventory_units": inventory,
            "defect_rate": defect_rate,
            "quality_yield": quality_yield,
            "utilization": utilization,
            "energy_per_good_unit_kwh": energy_per_good,
            "energy_cost_eur": cost_eur,
            "co2_kg": co2_kg,
            "service_level_cumulative": service_level_cum,
            "system_score": system_score,
            "raw_material_defect_extra": raw_material_defect_extra,
        }
        records.append(rec)

        prev_quality = quality
        prev_stability = stability
        prev_energy = energy_kwh
        prev_priority = priority

    return pd.DataFrame.from_records(records)


def summarize(df: pd.DataFrame) -> Dict[str, Any]:
    total_demand = df["demand_units"].sum()
    total_good = df["good_output_units_Q4_y"].sum()
    total_processed = df["processed_units_Q2_y"].sum()
    total_energy = df["energy_kwh_Q2_xy"].sum()
    return {
        "total_demand_units": total_demand,
        "total_processed_units": total_processed,
        "total_good_output_units": total_good,
        "final_backlog_units": df["backlog_units"].iloc[-1],
        "avg_shift_service_level": (df["good_output_units_Q4_y"] /
df["demand_units"]).clip(upper=1).mean(),
        "cumulative_service_level": total_good / max(total_demand, 1.0),
        "avg_quality_yield": df["quality_yield"].mean(),
        "total_energy_kwh": total_energy,
        "energy_per_good_unit_kwh": total_energy / max(total_good, 0.01),
        "total_energy_cost_eur": df["energy_cost_eur"].sum(),
        "total_co2_kg": df["co2_kg"].sum(),
        "avg_system_score": df["system_score"].mean(),
        "mode_counts": df["mode_Q1_z"].value_counts().to_dict(),
        "quality_counts": df["quality_Q4_z"].value_counts().to_dict(),
    }


def save_charts(df_mqm: pd.DataFrame, df_static: pd.DataFrame, out_dir: str) ->
List[str]:
    paths = []
    # Chart 1: demand/output/backlog.
    plt.figure(figsize=(10, 5))
    plt.plot(df_mqm["shift"], df_mqm["demand_units"], marker="o",
label="Nachfrage")
    plt.plot(df_mqm["shift"], df_mqm["good_output_units_Q4_y"], marker="o",
```

```
label="Guter Output MQM")
    plt.plot(df_static["shift"], df_static["good_output_units_Q4_y"],
marker="o", label="Guter Output statisch")
    plt.plot(df_mqm["shift"], df_mqm["backlog_units"], marker="o",
label="Rueckstand MQM")
    plt.xlabel("Schicht")
    plt.ylabel("Einheiten")
    plt.title("Nachfrage, Output und Rueckstand")
    plt.legend()
    plt.grid(True, alpha=0.3)
    p = os.path.join(out_dir, "chart_01_output_backlog.png")
    plt.tight_layout()
    plt.savefig(p, dpi=180)
    plt.close()
    paths.append(p)

    # Chart 2: energy and energy budget.
    plt.figure(figsize=(10, 5))
    plt.plot(df_mqm["shift"], df_mqm["energy_kwh_Q2_xy"], marker="o",
label="Energie MQM")
    plt.plot(df_static["shift"], df_static["energy_kwh_Q2_xy"], marker="o",
label="Energie statisch")
    plt.axhline(y=FactoryParams().energy_budget_kwh_per_shift, linestyle="--",
label="Budget")
    plt.xlabel("Schicht")
    plt.ylabel("kWh")
    plt.title("Energieverbrauch je Schicht")
    plt.legend()
    plt.grid(True, alpha=0.3)
    p = os.path.join(out_dir, "chart_02_energy.png")
    plt.tight_layout()
    plt.savefig(p, dpi=180)
    plt.close()
    paths.append(p)

    # Chart 3: energy per good unit.
    plt.figure(figsize=(10, 5))
    plt.plot(df_mqm["shift"], df_mqm["energy_per_good_unit_kwh"], marker="o",
label="MQM")
    plt.plot(df_static["shift"], df_static["energy_per_good_unit_kwh"],
marker="o", label="statisch")
    plt.xlabel("Schicht")
    plt.ylabel("kWh / gute Einheit")
    plt.title("Spezifischer Energieverbrauch")
    plt.legend()
    plt.grid(True, alpha=0.3)
    p = os.path.join(out_dir, "chart_03_specific_energy.png")
    plt.tight_layout()
    plt.savefig(p, dpi=180)
    plt.close()
    paths.append(p)

    # Chart 4: control mode as encoded categorical.
    mode_order = ["automatic", "rush", "energy_save", "stabilize",
```

```
"maintenance"]
    mode_map = {m: i for i, m in enumerate(mode_order)}
    plt.figure(figsize=(10, 4))
    plt.step(df_mqm["shift"], df_mqm["mode_Q1_z"].map(mode_map), where="mid",
label="Q1 Modus")
    plt.yticks(range(len(mode_order)), mode_order)
    plt.xlabel("Schicht")
    plt.ylabel("Q1_z")
    plt.title("Rueckkopplungsbasierte Steuerungsmodi")
    plt.grid(True, alpha=0.3)
    p = os.path.join(out_dir, "chart_04_modes.png")
    plt.tight_layout()
    plt.savefig(p, dpi=180)
    plt.close()
    paths.append(p)

    return paths


def fmt(x: Any, nd: int = 2) -> str:
    if isinstance(x, (float, np.floating)):
        return f"{x:,.{nd}f}".replace(",", "X").replace(".", ",").replace("X",
".")
    return str(x)


def add_page_number(canvas, doc):
    canvas.saveState()
    canvas.setFont("Helvetica", 8)
    canvas.drawRightString(A4[0] - 1.5 * cm, 1.0 * cm, f"Seite {doc.page}")
    canvas.restoreState()


def make_report(params: FactoryParams, df_mqm: pd.DataFrame, df_static:
pd.DataFrame,
                charts: List[str], out_pdf: str) -> None:
    styles = getSampleStyleSheet()
    styles.add(ParagraphStyle(name="TitleCenter", parent=styles["Title"],
alignment=TA_CENTER, fontSize=18, leading=22, spaceAfter=12))
    styles.add(ParagraphStyle(name="Small", parent=styles["BodyText"],
fontSize=8, leading=10))
    styles.add(ParagraphStyle(name="Caption", parent=styles["BodyText"],
fontSize=8, leading=10, italic=True, textColor=colors.HexColor("#444444")))
    styles.add(ParagraphStyle(name="H2", parent=styles["Heading2"], fontSize=13,
leading=16, spaceBefore=8, spaceAfter=6))
    styles.add(ParagraphStyle(name="TableCell", parent=styles["BodyText"],
fontSize=7.2, leading=8.4))
    styles.add(ParagraphStyle(name="TableHead", parent=styles["BodyText"],
fontSize=7.2, leading=8.4, fontName="Helvetica-Bold"))

    def wrap_rows(rows):
        wrapped = []
        for ridx, row in enumerate(rows):
            style = styles["TableHead"] if ridx == 0 else styles["TableCell"]
```

```
            wrapped.append([Paragraph(str(cell), style) for cell in row])
        return wrapped

    doc = SimpleDocTemplate(out_pdf, pagesize=A4, rightMargin=1.4*cm,
leftMargin=1.4*cm, topMargin=1.4*cm, bottomMargin=1.4*cm)
    story = []

    summary_mqm = summarize(df_mqm)
    summary_static = summarize(df_static)

    story.append(Paragraph("Konkrete MQM-Simulation einer fiktiven virtuellen
Fabrik", styles["TitleCenter"]))
    story.append(Paragraph("Anwendungsfall: VF-MQM-01 - Produktionszelle fuer
Aluminium-Pumpengehaeuse", styles["BodyText"]))
    story.append(Spacer(1, 0.2*cm))
    story.append(Paragraph("Kurzfassung", styles["H2"]))
    story.append(Paragraph(
        "Dieser Bericht beschreibt eine reproduzierbare diskrete Simulation
einer fiktiven virtuellen Fabrik. "
        "Die Fabrik ist fiktiv, die technischen Eckwerte sind jedoch realistisch
parametrisiert: zwei CNC-Bearbeitungszentren "
        "mit VF-2-aehnlicher Leistung, ein UR10e-aehnlicher Cobot, deutsche
Strompreis- und CO2-Faktoren. "
        "Die Simulation bildet Q3 Planung, Q1 Steuerung, Q2 Prozess und Q4
Ergebnis als Teilstrukturen eines Systems ab. "
        "Zustaende sind typisiert: numerisch, ordinal, nominal und relational.",
styles["BodyText"]))

    story.append(Paragraph("1. Reale Parameter und Quellen", styles["H2"]))
    param_data = [
        ["Parameter", "Wert in der Simulation", "Begruendung / Quelle"],
        ["CNC-Spindelleistung", f"{params.cnc_spindle_nominal_kw} kW nominal;
{params.cnc_run_kw} kW mittlere Last", "VF-2: 30 hp / 22,4 kW, 8100 rpm;
mittlere Last als Modellannahme"],
        ["CNC-Kuehlmittelpumpe", f"{params.coolant_pump_kw_per_machine} kW je
Maschine", "VF-2-Spezifikation: 0,6 kW Pumpe"],
        ["Cobot-Leistung", f"{params.cobot_average_kw} kW", "UR10e: 615 W
average, ca. 350 W typical program"],
        ["Strompreis", f"{params.electricity_price_eur_per_kwh} EUR/kWh",
"Deutschland, Nicht-Haushalt, Jun 2025, Eurostat/CEIC"],
        ["CO2-Faktor Strom", f"{params.co2_kg_per_kwh} kg CO2/kWh", "UBA: 363 g
CO2/kWh fuer deutschen Strommix 2024"],
        ["Schichtmodell", f"{params.n_shifts} Schichten x {params.shift_hours}
h", "Modellannahme"],
        ["Taktzeit automatisch", f"{params.cycle_time_min_auto} min/Stueck",
"Plausible Prozessannahme fuer kleines Fraesteil"],
    ]
    t = Table(wrap_rows(param_data), colWidths=[4.0*cm, 4.5*cm, 8.0*cm])
    t.setStyle(TableStyle([
        ("BACKGROUND", (0,0), (-1,0), colors.HexColor("#EAEAEA")),
        ("GRID", (0,0), (-1,-1), 0.3, colors.grey),
        ("VALIGN", (0,0), (-1,-1), "TOP"),
        ("FONTNAME", (0,0), (-1,0), "Helvetica-Bold"),
        ("FONTSIZE", (0,0), (-1,-1), 8),
```

```
    ]))
    story.append(t)
    story.append(Spacer(1, 0.2*cm))
    story.append(Paragraph(
        "Hinweis: Die reale Maximalleistung einer Spindel ist nicht gleich dem
durchschnittlichen Prozessverbrauch. "
        "Darum wird die 22,4-kW-Angabe als technische Obergrenze verwendet und
fuer den laufenden Prozess eine mittlere "
        "Last von 11,2 kW je CNC angesetzt.", styles["Caption"]))

    story.append(Paragraph("2. MQM-Abbildung", styles["H2"]))
    mqm_data = [
        ["Quadrant", "Rolle in der Fabrik", "Beispielzustand"],
        ["Q3(S) Planung", "Soll-Menge, Prioritaet, Sequenzregel, Portfolio",
"Q3_y = geplante Einheiten; Q3_z = Prioritaet"],
        ["Q1(S) Steuerung", "Moduswahl, Kapazitaet, Regelstrategie,
Maschinenkonfiguration", "Q1_z =
automatic/rush/energy_save/stabilize/maintenance"],
        ["Q2(S) Prozess", "tatsaechliche Bearbeitung, Energie, Stabilitaet",
"Q2_xy = kWh; Q2_yz = stable/strained/unstable"],
        ["Q4(S) Ergebnis", "Guter Output, Qualitaet, Rueckstand, Kosten, CO2",
"Q4_y = gute Einheiten; Q4_z = high/medium/low"],
    ]
    t = Table(wrap_rows(mqm_data), colWidths=[3.0*cm, 6.5*cm, 7.0*cm])
    t.setStyle(TableStyle([
        ("BACKGROUND", (0,0), (-1,0), colors.HexColor("#EAEAEA")),
        ("GRID", (0,0), (-1,-1), 0.3, colors.grey),
        ("VALIGN", (0,0), (-1,-1), "TOP"),
        ("FONTNAME", (0,0), (-1,0), "Helvetica-Bold"),
        ("FONTSIZE", (0,0), (-1,-1), 8),
    ]))
    story.append(t)

    story.append(Paragraph("3. Rueckkopplungslogik", styles["H2"]))
    story.append(Paragraph(
        "Die Simulation verwendet eine zeitdiskrete Hauptschleife: Q4(t) und
Q2(t) melden Output, Qualitaet, Energie und "
        "Stabilitaet zurueck; daraus erzeugt Q3(t+1) einen aktualisierten Plan;
Q1(t+1) loest Zielkonflikte; Q2(t+1) fuehrt "
        "den Prozess aus; Q4(t+1) bewertet Ergebnis und KPI. Die Konfliktregel
lautet: Stabilitaet und Qualitaet vor Energie, "
        "Energie vor Liefertermin, Liefertermin vor Durchsatz.",
styles["BodyText"]))
    rules = [
        ["Feedback", "Regel"],
        ["Rueckstand hoch", "Q3_z eskaliert auf high/critical; Q1 kann rush
waehlen"],
        ["Qualitaet low", "Q1_z wird stabilize"],
        ["Energie > 110% Budget", "Q1_z wird energy_save"],
        ["Schicht 20", "Praeventive Wartung: Q1_z wird maintenance"],
    ]
    t = Table(wrap_rows(rules), colWidths=[5.0*cm, 11.5*cm])
    t.setStyle(TableStyle([
        ("BACKGROUND", (0,0), (-1,0), colors.HexColor("#EAEAEA")),
```

```
        ("GRID", (0,0), (-1,-1), 0.3, colors.grey),
        ("FONTNAME", (0,0), (-1,0), "Helvetica-Bold"),
        ("FONTSIZE", (0,0), (-1,-1), 8),
    ]))
    story.append(t)

    story.append(PageBreak())
    story.append(Paragraph("4. Simulationsergebnisse", styles["H2"]))
    comp = [
        ["Kennzahl", "MQM-Regelung", "Statische Regelung", "Differenz MQM -
statisch"],
        ["Nachfrage gesamt [Stueck]", fmt(summary_mqm["total_demand_units"], 0),
fmt(summary_static["total_demand_units"], 0), "0"],
        ["Guter Output [Stueck]", fmt(summary_mqm["total_good_output_units"],
1), fmt(summary_static["total_good_output_units"], 1),
fmt(summary_mqm["total_good_output_units"] -
summary_static["total_good_output_units"], 1)],
        ["Finaler Rueckstand [Stueck]", fmt(summary_mqm["final_backlog_units"],
1), fmt(summary_static["final_backlog_units"], 1),
fmt(summary_mqm["final_backlog_units"] - summary_static["final_backlog_units"],
1)],
        ["Kumulativer Servicegrad",
fmt(summary_mqm["cumulative_service_level"]*100, 1)+"%",
fmt(summary_static["cumulative_service_level"]*100, 1)+"%",
fmt((summary_mqm["cumulative_service_level"]-summary_static["cumulative_service_
level"])*100, 1)+" pp"],
        ["Qualitaetsausbeute avg.", fmt(summary_mqm["avg_quality_yield"]*100,
2)+"%", fmt(summary_static["avg_quality_yield"]*100, 2)+"%",
fmt((summary_mqm["avg_quality_yield"]-summary_static["avg_quality_yield"])*100,
2)+" pp"],
        ["Energie gesamt [kWh]", fmt(summary_mqm["total_energy_kwh"], 1),
fmt(summary_static["total_energy_kwh"], 1), fmt(summary_mqm["total_energy_kwh"]
- summary_static["total_energy_kwh"], 1)],
        ["kWh / gute Einheit", fmt(summary_mqm["energy_per_good_unit_kwh"], 2),
fmt(summary_static["energy_per_good_unit_kwh"], 2),
fmt(summary_mqm["energy_per_good_unit_kwh"] -
summary_static["energy_per_good_unit_kwh"], 2)],
        ["Stromkosten [EUR]", fmt(summary_mqm["total_energy_cost_eur"], 0),
fmt(summary_static["total_energy_cost_eur"], 0),
fmt(summary_mqm["total_energy_cost_eur"] -
summary_static["total_energy_cost_eur"], 0)],
        ["CO2 [kg]", fmt(summary_mqm["total_co2_kg"], 0),
fmt(summary_static["total_co2_kg"], 0), fmt(summary_mqm["total_co2_kg"] -
summary_static["total_co2_kg"], 0)],
        ["Systemscore avg.", fmt(summary_mqm["avg_system_score"], 1),
fmt(summary_static["avg_system_score"], 1), fmt(summary_mqm["avg_system_score"]
- summary_static["avg_system_score"], 1)],
    ]
    t = Table(wrap_rows(comp), colWidths=[5.2*cm, 3.6*cm, 3.6*cm, 4.1*cm])
    t.setStyle(TableStyle([
        ("BACKGROUND", (0,0), (-1,0), colors.HexColor("#EAEAEA")),
        ("GRID", (0,0), (-1,-1), 0.3, colors.grey),
        ("FONTNAME", (0,0), (-1,0), "Helvetica-Bold"),
        ("FONTSIZE", (0,0), (-1,-1), 8),
```

```
        ("ALIGN", (1,1), (-1,-1), "RIGHT"),
    ]))
    story.append(t)
    story.append(Spacer(1, 0.2*cm))
    story.append(Paragraph(
        "Interpretation: Die MQM-Regelung greift aktiv in Q1 ein. Dadurch steigt
in Spitzenphasen der Output, waehrend "
        "Qualitaets- und Energiegrenzen ueber Rueckkopplungen begrenzt werden.
Der statische Referenzfall bleibt in automatic "
        "und kann Demand-Spikes schlechter kompensieren.", styles["BodyText"]))

    for i, ch in enumerate(charts[:2], start=1):
        story.append(Spacer(1, 0.2*cm))
        story.append(Image(ch, width=16.5*cm, height=8.2*cm))
        story.append(Paragraph(f"Abbildung {i}: Simulationsergebnis, Quelle:
eigene Simulation.", styles["Caption"]))

    story.append(PageBreak())
    for i, ch in enumerate(charts[2:], start=3):
        story.append(Spacer(1, 0.2*cm))
        story.append(Image(ch, width=16.5*cm, height=8.2*cm))
        story.append(Paragraph(f"Abbildung {i}: Simulationsergebnis, Quelle:
eigene Simulation.", styles["Caption"]))

    story.append(Paragraph("5. Auszug aus dem MQM-Zustandsraum", styles["H2"]))
    sample = df_mqm[[
        "shift", "planned_units_Q3_y", "priority_Q3_z", "load_Q3_xy",
"mode_Q1_z",
        "processed_units_Q2_y", "energy_kwh_Q2_xy", "stability_Q2_yz",
"good_output_units_Q4_y",
        "quality_Q4_z", "backlog_units"
    ]].head(12).copy()
    table_data = [["Schicht", "Q3_y Plan", "Q3_z", "Q3_xy", "Q1_z", "Q2_y",
"Q2_xy kWh", "Q2_yz", "Q4_y", "Q4_z", "Backlog"]]
    for _, r in sample.iterrows():
        table_data.append([
            int(r["shift"]), fmt(r["planned_units_Q3_y"], 1),
r["priority_Q3_z"], fmt(r["load_Q3_xy"], 2), r["mode_Q1_z"],
            fmt(r["processed_units_Q2_y"], 1), fmt(r["energy_kwh_Q2_xy"], 1),
r["stability_Q2_yz"],
            fmt(r["good_output_units_Q4_y"], 1), r["quality_Q4_z"],
fmt(r["backlog_units"], 1)
        ])
    t = Table(wrap_rows(table_data), colWidths=[1.2*cm, 1.6*cm, 1.6*cm, 1.5*cm,
2.1*cm, 1.5*cm, 1.9*cm, 1.8*cm, 1.5*cm, 1.4*cm, 1.6*cm])
    t.setStyle(TableStyle([
        ("BACKGROUND", (0,0), (-1,0), colors.HexColor("#EAEAEA")),
        ("GRID", (0,0), (-1,-1), 0.25, colors.grey),
        ("FONTNAME", (0,0), (-1,0), "Helvetica-Bold"),
        ("FONTSIZE", (0,0), (-1,-1), 6.7),
        ("ALIGN", (1,1), (-1,-1), "RIGHT"),
        ("ALIGN", (2,1), (4,-1), "LEFT"),
    ]))
    story.append(t)
```

```
    story.append(Paragraph("6. Reproduzierbarkeit", styles["H2"]))
    story.append(Paragraph(
        "Die Simulation ist deterministisch reproduzierbar. Alle
Eingangsparameter stehen in der JSON-Datei; alle "
        "Schichtergebnisse stehen in der CSV-Datei; die Python-Datei enthaelt
die vollstaendige Simulationslogik. "
        "Die wichtigste methodische Einschraenkung ist, dass keine realen
Betriebsdaten einer konkreten Fabrik kalibriert wurden. "
        "Das Ergebnis ist daher ein plausibler digitaler Demonstrator, kein
validierter industrieller Digital Twin.", styles["BodyText"]))

    story.append(Paragraph("7. Quellenhinweise", styles["H2"]))
    sources = [
        "Haas VF-2 Spezifikation: 30 hp / 22,4 kW Spindel, 0,6 kW
Kuehlmittelpumpe.",
        "Universal Robots UR10e technische Spezifikation: 615 W average power
consumption, ca. 350 W typical program.",
        "Umweltbundesamt: 363 g CO2/kWh deutscher Strommix 2024.",
        "Eurostat/CEIC: Deutschland Nicht-Haushalt Strompreis Jun 2025; in der
Simulation 0,241 EUR/kWh.",
        "MQM-Modellgrundlage: Quadranten als Teilstrukturen; x, y, z sowie xy,
xz, yz als typisierte Zustandskomponenten.",
    ]
    for s in sources:
        story.append(Paragraph("- " + s, styles["Small"]))

    doc.build(story, onFirstPage=add_page_number, onLaterPages=add_page_number)


def main() -> None:
    out_dir = os.path.dirname(os.path.abspath(__file__))
    params = FactoryParams()
    df_mqm = simulate("mqm", params)
    df_static = simulate("static", params)

    df_all = pd.concat([df_mqm, df_static], ignore_index=True)
    csv_path = os.path.join(out_dir, "mqm_factory_simulation_results.csv")
    df_all.to_csv(csv_path, index=False)

    params_path = os.path.join(out_dir, "mqm_factory_simulation_params.json")
    with open(params_path, "w", encoding="utf-8") as f:
        json.dump(asdict(params), f, indent=2, ensure_ascii=False)

    charts = save_charts(df_mqm, df_static, out_dir)

    summary = {
        "mqm": summarize(df_mqm),
        "static": summarize(df_static),
    }
    with open(os.path.join(out_dir, "mqm_factory_simulation_summary.json"), "w",
encoding="utf-8") as f:
        json.dump(summary, f, indent=2, ensure_ascii=False)
```

```
    pdf_path = os.path.join(out_dir, "mqm_factory_simulation_report.pdf")
    make_report(params, df_mqm, df_static, charts, pdf_path)

    print("created", csv_path)
    print("created", params_path)
    print("created", pdf_path)
    for ch in charts:
        print("created", ch)


if __name__ == "__main__":
    main()
```