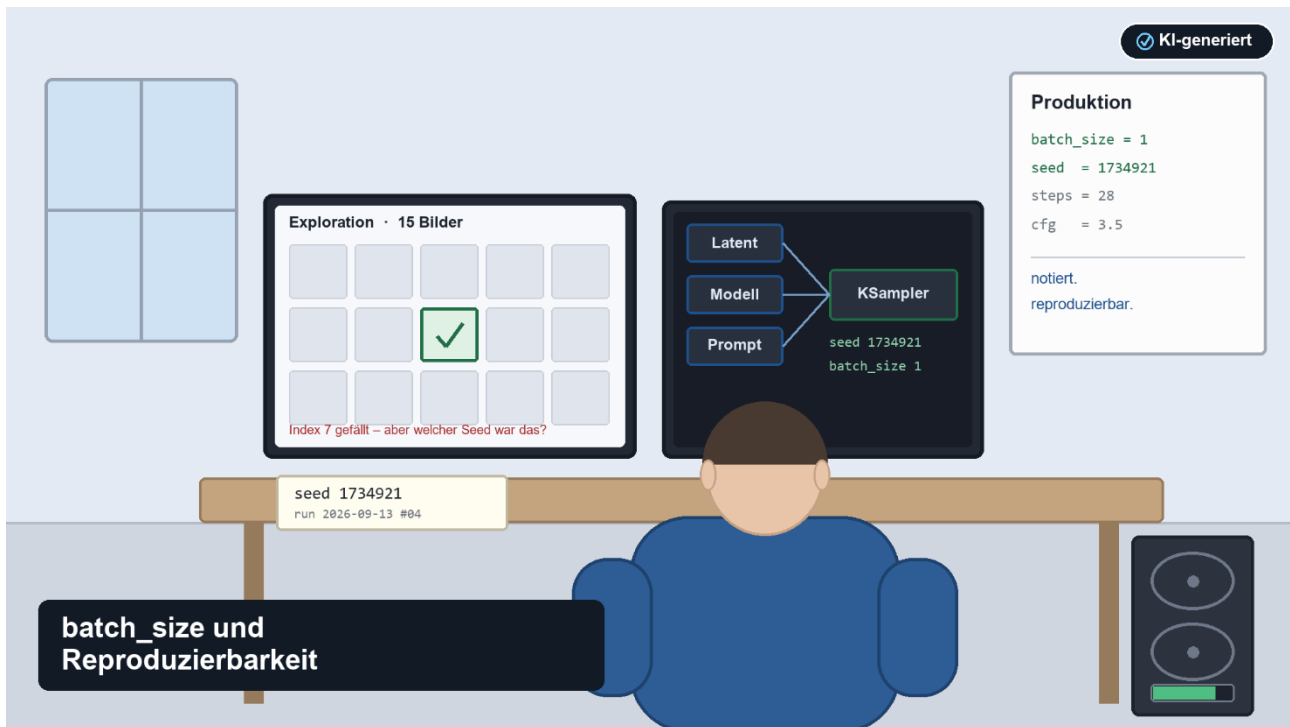


batch_size und Reproduzierbarkeit



Exploration links, Produktion rechts: Der Stapel liefert Kandidaten, der notierte Seed liefert Wiederholbarkeit. Bild selbst erzeugt und als KI-generiert gekennzeichnet.

Wissen	Beratung	Schulung
Alle Grundlagen zur KI-Bildgenerierung an einer Stelle – vom Diffusionsprinzip bis zur Governance.	Von der Testinstallation zur belastbaren Bildproduktion mit Nachweis und klaren Zuständigkeiten.	Dein Team lernt die Regler zu bedienen – und zu protokollieren, was es eingestellt hat.
KI-Bildgenerierung	KI-Bildgenerierung Beratung	KI-Bildgenerierung Schulung

Ein Schieberegler, der statt einem Bild gleich acht ausspuckt, ist die verführerischste Einstellung der ganzen Oberfläche. Mehr Auswahl, weniger Warten, und das Gefühl, dass die teure Grafikkarte endlich mal arbeitet. Für die ersten zwei Stunden eines Projekts ist das auch völlig richtig. Nur wird aus diesen zwei Stunden in fast jedem Unternehmen irgendwann ein Freigabeprozess, eine Kampagne und ein Motiv, das sechs Wochen später noch einmal gebraucht wird – im Querformat, mit anderem Text, ansonsten aber bitte genau so.

Bei einem Maschinenbauer habe ich genau diesen Moment miterlebt. Das Marketing hatte Motive für eine Messe erzeugt, 64 Bilder in einem Rutsch, ein Volltreffer war dabei. Notiert wurde der Start-Seed des Stapels, weil in der Oberfläche eben nur der eine sichtbar war. Sechs Wochen später sollte dasselbe Motiv als Rollup nachgeliefert werden. Zwischenzeitlich war ComfyUI aktualisiert worden, das Attention-Backend hatte gewechselt, und der Rechner, auf dem das Original entstanden war, hatte inzwischen eine andere Grafikkarte. Der notierte Seed lieferte ein Bild. Nur eben nicht dieses. Am Ende standen zwei Tage Nacharbeit für etwas, das mit einer Zeile im Protokoll erledigt gewesen wäre.

Genau darum geht es hier. batch_size ist kein Qualitätsregler, sondern ein Mengen- und Tempo-Regler mit einem unangenehmen Nebeneffekt: Er entscheidet mit darüber, ob du ein Bild später

wiederfindest. Dieser Beitrag klärt, was batch_size technisch tut, warum es nicht dasselbe ist wie batch_count, wie die Werkzeuge den Seed im Stapel vergeben – und warum die Antwort in ComfyUI eine ganz andere ist als in der WebUI-Familie. Wenn du das Fundament dahinter noch nicht kennst, hilft [Basiswissen KI-Bildgenerierung: Wie Diffusion funktioniert](#) als Einstieg.

Was du aus diesem Beitrag mitnimmst

batch_size rechnet mehrere Bilder gleichzeitig in einem Durchgang durch das Modell. batch_count rechnet sie nacheinander. Die Bildmenge ist dieselbe, das Speicherprofil ist es nicht.

Die Werkzeuge vergeben den Seed im Stapel unterschiedlich. In der WebUI-Familie bekommt jedes Bild eine eigene, fortlaufend hochgezählte Seed-Nummer. In ComfyUI gilt ein Seed für den gesamten Stapel, und die einzelnen Bilder haben gar keine eigene Nummer – sie haben einen Index.

Ein Stapel liefert nicht garantiert dieselben Bilder wie Einzelläufe mit denselben Werten. Die Reihenfolge der Rechenoperationen hängt von der Stapelgröße ab, und das kann das Ergebnis verändern. Wer bit-genaue Wiederholbarkeit braucht, arbeitet mit batch_size 1.

Die Seed-Nummer ist nur zusammen mit Modellversion, Software, Einstellungen und Hardware ein Versprechen. Alles davon gehört ins Protokoll, nicht nur die Zahl.

Der Tempogewinn großer Stapel läuft aus, der Speicherbedarf nicht. Ein mittlerer Wert ist in der Regel der beste Kompromiss.

batch_size und batch_count – zwei Regler, die gern verwechselt werden

Die beiden Regler stehen in den meisten Oberflächen direkt untereinander, heißen fast gleich und tun etwas grundsätzlich Verschiedenes. Wer sie verwechselt, wundert sich entweder über Abbrüche oder über verschenkte Rechenzeit.

batch_size: nebeneinander rechnen

batch_size ist die Anzahl der Bilder, die in einem einzigen Durchgang gleichzeitig durch das Modell geschoben werden. Technisch werden die Bilder zu einem einzigen Tensor gestapelt – daher der Name. Das Modell verarbeitet diesen Stapel in jedem Rechenschritt gemeinsam. Der Vorteil: Eine moderne Grafikkarte ist bei einem einzelnen Bild oft gar nicht ausgelastet, weil sie zwischendurch auf Daten wartet. Ein Stapel füllt diese Lücken und liefert mehr Bilder pro Minute.

Der Preis steht im Grafikspeicher. Die Modellgewichte liegen nur einmal im VRAM, egal wie groß der Stapel ist. Aber die Zwischenergebnisse – die Aktivierungen – fallen für jedes Bild im Stapel separat an und wachsen linear mit. Reicht der Speicher nicht, bricht der Lauf mit der berüchtigten Meldung „out of memory“ ab. Und zwar nicht nach dem dritten Bild, sondern mitten im Durchgang: Der gesamte Stapel ist dann verloren, nicht nur das Bild, das nicht mehr gepasst hätte.

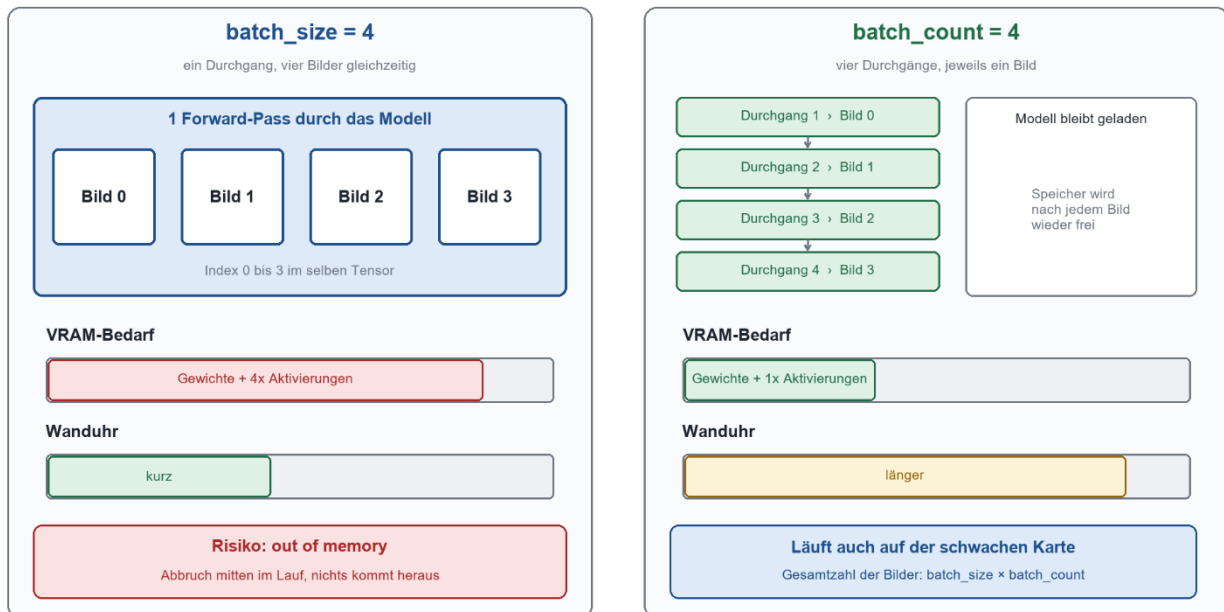
batch_count: hintereinander rechnen

batch_count – in ComfyUI heißt das Gegenstück schlicht, wie oft du den Auftrag in die Warteschlange stellst – bestimmt, wie viele Durchgänge nacheinander gerechnet werden. Jeder Durchgang belegt nur so viel Speicher wie ein einzelner Stapel, und danach wird der Arbeitsspeicher der Grafikkarte wieder frei. Das Modell bleibt geladen, es muss also nicht jedes Mal neu eingelesen werden. Das Ergebnis dauert länger, riskiert aber keinen Abbruch, solange ein einzelner Durchgang passt.

Beide Regler lassen sich kombinieren, und die Gesamtmenge ist schlicht das Produkt: batch_size 2 mit batch_count 4 ergibt acht Bilder in vier Durchgängen zu je zwei parallel gerechneten. Die alte Faustregel lautet: batch_size so hoch, wie der Speicher es klaglos mitmacht, den Rest über

batch_count. Diese Regel ist gut für den Durchsatz und schlecht für die Nachvollziehbarkeit – dazu gleich mehr.

batch_size rechnet nebeneinander, batch_count hintereinander



Gleiche Bildmenge, völlig unterschiedliches Speicherprofil – und unterschiedliche Fallhöhe.

Skizze 1: Dieselbe Bildmenge, zwei Wege. batch_size füllt die Grafikkarte und riskiert den Abbruch, batch_count nimmt sich Zeit und bleibt im Speicherbudget.

Merkmal	batch_size	batch_count
Rechenweise	alle Bilder gleichzeitig in einem Durchgang	ein Durchgang nach dem anderen
VRAM-Bedarf	steigt mit jedem zusätzlichen Bild im Stapel	konstant, entspricht einem Durchgang
Tempo	höher, bis das Rechenwerk ausgelastet ist	niedriger, skaliert linear mit der Anzahl
Verhalten bei Speichermangel	Abbruch, der komplette Stapel geht verloren	unkritisch, solange ein Durchgang passt
Zwischenergebnisse sichtbar	erst am Ende des Durchgangs, alle auf einmal	nach jedem Durchgang, du kannst früh abbrechen
Seed-Vergabe	je nach Werkzeug fortlaufend oder gar keine eigene Nummer	in der Regel ein sauber hochgezählter Seed je Durchgang
Empfehlung bei wenig VRAM	auf 1 lassen	hochdrehen, kostet nur Zeit
Empfehlung in der Produktion	auf 1 lassen	je nach Bedarf, mit festem Seed pro Lauf

Der Unterschied zwischen Auslastung und Nutzen

Eine Grafikkarte, die bei batch_size 8 zu 99 Prozent ausgelastet ist, sieht im Monitoring beeindruckend aus. Für das Projekt zählt aber nicht die Auslastung, sondern wie viele brauchbare und wiederauffindbare Bilder am Ende herauskommen. Sechzig unsortierte Bilder ohne Zuordnung sind weniger wert als drei Bilder mit vollständigem Protokoll.

Der Seed im Stapel – und warum die Antwort vom Werkzeug abhängt

Der Seed ist die Zahl, aus der das Startrauschen eines Bildes abgeleitet wird. Gleicher Seed, gleiche Einstellungen, gleiches Bild – so lautet das Versprechen, und in der Einzelbild-Welt hält es auch weitgehend. Wie das im Detail funktioniert, erklärt [Der Seed bei der KI-Bildgenerierung](#). Sobald du aber mehrere Bilder gleichzeitig rechnest, stellt sich eine Frage, die kaum eine Oberfläche ehrlich beantwortet: Welches Bild hat welchen Seed?

Die WebUI-Familie zählt hoch

In der Automatic1111-Linie und ihren Ablegern – Forge, reForge und was sonst noch von diesem Codestamm abstammt – ist die Antwort einfach. Die Software legt vor dem Lauf eine Liste aller Seeds an, indem sie den Start-Seed fortlaufend hochzählt: Das erste Bild bekommt deinen Seed, das zweite den Seed plus eins, und so weiter. Diese Zählung läuft über `batch_size` und `batch_count` gemeinsam durch. Ein Lauf mit `batch_size` 2 und `batch_count` 3 erzeugt also sechs Bilder mit sechs aufeinanderfolgenden Seed-Nummern.

Das Gute daran: Der jeweilige Einzelseed landet im Parameterblock, den die Software in die PNG-Datei schreibt. Du kannst ihn also nachträglich aus dem Bild auslesen, auch wenn die Oberfläche nur den Start-Seed anzeigt. Die Ausnahme ist der Variations-Seed: Sobald du mit Subseed und Variationsstärke arbeitest, bleibt der Haupt-Seed über den ganzen Lauf konstant und stattdessen wird der Variations-Seed hochgezählt. Wer das nicht weiß, notiert die falsche Zahl.

Ein Hinweis zum Zustand dieser Werkzeugfamilie, weil er die Praxis betrifft: Die Ursprungsversion von Automatic1111 hat seit Version 1.10.1 aus dem Sommer 2024 keine neue Veröffentlichung mehr gesehen, der Entwicklungszweig bekommt nur noch vereinzelt Änderungen. Das ursprüngliche Forge-Projekt ist seit Mitte 2025 nicht mehr weiterentwickelt worden; wer in dieser Familie bleiben will, landet heute meist bei reForge. Für moderne Modelle wie die FLUX-Generation ist das ein handfestes Problem – was auch der Grund ist, warum die meisten ernsthaften Produktionsumgebungen inzwischen auf ComfyUI stehen.

ComfyUI zählt nicht hoch – es zieht weiter

Hier wird es interessant, und hier liegt der Irrtum, der die meisten Suchen nach dem verlorenen Bild scheitern lässt. ComfyUI vergibt im Stapel keine Einzelseeds. Der Sampler setzt den Zufallsgenerator einmal mit deinem Seed auf und zieht dann in einem Zug so viel Rauschen, wie der gesamte Stapel braucht. Das Rauschen für Bild 0 stammt aus dem Anfang dieses Stroms, das für Bild 1 aus der unmittelbaren Fortsetzung, und so weiter.

Daraus folgen zwei Dinge, die du kennen musst. Erstens: Das Bild an Index 0 eines Stapels startet mit demselben Rauschen wie ein Einzellauf mit demselben Seed. Zweitens: Für die Bilder an Index 1 und aufwärts gibt es keine Seed-Nummer, die du irgendwo eintragen könntest. Seed plus eins liefert dir etwas völlig anderes – nämlich den Anfang eines neuen Rauschstroms, nicht dessen Fortsetzung. Genau hier scheitern die Rekonstruktionsversuche reihenweise, weil Leute die Logik der WebUI-Familie auf ComfyUI übertragen.

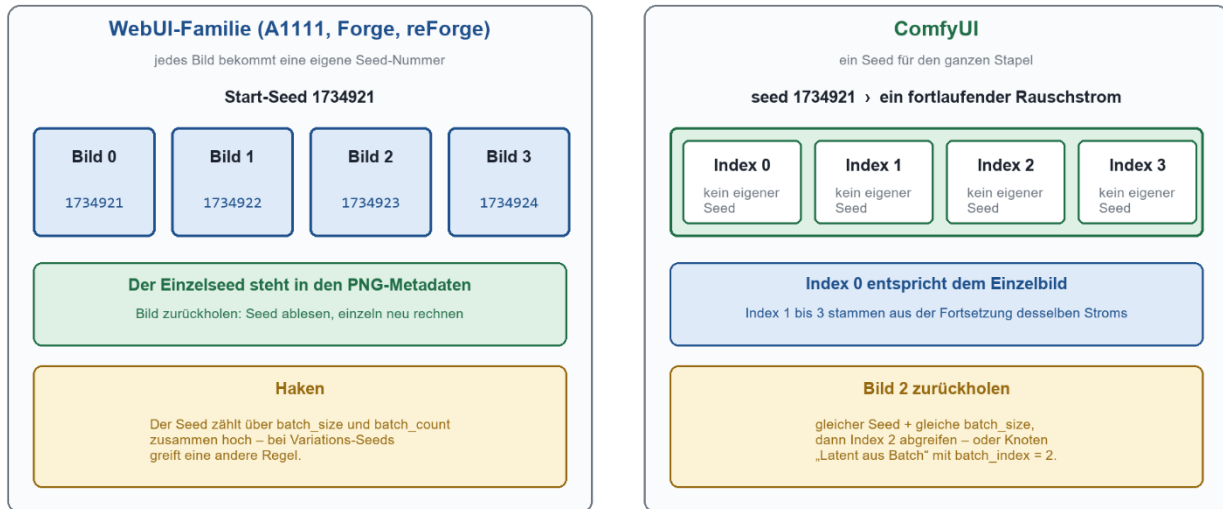
Es gibt zwei saubere Wege, ein bestimmtes Bild aus einem ComfyUI-Stapel wiederzuholen. Der eine: denselben Seed und dieselbe `batch_size` einstellen, den Lauf wiederholen und das Bild an derselben Indexposition nehmen. Der andere, elegantere: den Knoten „Latent aus Batch“ verwenden, der einen Batch-Index mitführt. Der Sampler überspringt dann gezielt die Rauschziehungen der vorangehenden Indizes und trifft dieselbe Stelle im Strom, ohne den ganzen Stapel rechnen zu müssen. Beides setzt voraus, dass du weißt, welcher Index es war – und genau deshalb muss der Index mit ins Protokoll.

Der Klassiker: Seed plus eins funktioniert in ComfyUI nicht

Wenn dir im ComfyUI-Stapel das dritte Bild gefällt, ist sein Seed nicht „Start-Seed plus zwei“. Es gibt keinen eigenen Seed. Wer diese Zahl notiert und später einträgt, bekommt ein Bild, das mit dem gesuchten nichts zu tun hat – und sucht dann stundenlang den Fehler bei Prompt, Sampler oder Modell.

Notiere stattdessen: Seed des Laufs, batch_size des Laufs und die Indexposition. Diese drei Angaben zusammen sind die Adresse des Bildes.

Wo der Seed im Stapel landet – zwei Werkzeuge, zwei Antworten



Gilt in beiden Welten

Die Seed-Nummer allein ist kein Versprechen. Sie gilt nur zusammen mit derselben batch_size, demselben Modell, derselben Software und derselben Grafikkarte. Ändert sich eine dieser Größen, ändert sich das Bild – manchmal unsichtbar, manchmal deutlich.

Skizze 2: Links vergibt die Software je Bild eine eigene Seed-Nummer, rechts gilt ein Seed für den ganzen Stapel und die Bilder unterscheiden sich nur durch ihre Position.

Umgebung	Seed-Vergabe im Stapel	So holst du ein Einzelbild zurück
Automatic1111, Forge, reForge	fortlaufend hochgezählt über batch_size und batch_count hinweg	Einzelseed aus den PNG-Metadaten lesen, mit batch_size 1 neu rechnen
Automatic1111 mit Variations-Seed	Haupt-Seed bleibt fest, der Variations-Seed zählt hoch	Haupt-Seed, Variations-Seed und Variationsstärke gemeinsam notieren
ComfyUI, Standard-Sampler	ein Seed für den gesamten Stapel, Bilder haben nur einen Index	gleicher Seed plus gleiche batch_size, dann denselben Index abgreifen
ComfyUI mit Knoten „Latent aus Batch“	Batch-Index wird bis in die Rauscherzeugung durchgereicht	Seed eintragen, batch_index setzen, einzeln rechnen
Cloud-Dienste mit fester Oberfläche	je nach Anbieter, oft gar nicht offengelegt	nur über die vom Anbieter gespeicherte Lauf-Historie, sofern vorhanden

Die letzte Zeile ist der Grund, warum die Frage nach der Reproduzierbarkeit sehr schnell zur Architekturfrage wird. Wenn dein Anbieter nicht offenlegt, wie er den Seed im Stapel vergibt, hast du das Nachweisproblem an einen Dritten ausgelagert, ohne es zu lösen. Der Vergleich der beiden Betriebsmodelle steht in [Cloud vs. lokale KI-Bildgenerierung](#).

Die unbequeme Wahrheit: ein Stapel rechnet nicht identisch

Jetzt kommt der Teil, den die meisten Einführungstexte weglassen, weil er unhandlich ist. Die gängige Behauptung lautet: Ob ein Bild allein oder als Teil eines Stapels entsteht, ändert nichts – bei gleichem Seed und gleichen Einstellungen komme exakt dasselbe Bild heraus. Das ist als Faustregel brauchbar und als technische Aussage falsch.

Warum die Stapelgröße das Ergebnis verändern kann

Ein Diffusionsmodell rechnet mit Fließkommazahlen begrenzter Genauigkeit. Fließkomma-Addition ist nicht assoziativ: Die Reihenfolge, in der Zwischenergebnisse zusammengefasst werden, beeinflusst die letzten Nachkommastellen. Genau diese Reihenfolge ändert sich mit der Stapelgröße. Bibliotheken wählen für unterschiedliche Tensorformen unterschiedliche, jeweils optimierte Rechenkerne; Aufmerksamkeits- und Normalisierungsschichten fassen über andere Achsen zusammen. Das Ergebnis weicht zunächst in der zwölften Nachkommastelle ab.

Das Problem ist, dass ein Diffusionsprozess diese Abweichung über die Schritte verstärkt. Jeder Sampling-Schritt nimmt das Ergebnis des vorigen als Eingabe. Eine winzige Differenz im ersten Schritt ist nach zwanzig Schritten kein Rauschen mehr, sondern kann eine andere Bildkomposition bedeuten. In der Praxis erlebst du beide Extreme: Meist sind die Bilder ununterscheidbar, gelegentlich dreht sich ein Gesicht, ändert sich ein Schriftzug oder verschwindet ein Detail. Dass in der Automatic1111-Entwicklung seit Jahren ein offener Fehlerbericht zu genau diesem Thema existiert – Ergebnisse sind über unterschiedliche Stapelgrößen hinweg nicht deterministisch – zeigt, dass das kein Randphänomen ist.

Empfindlich ist das vor allem dort, wo das Bild ohnehin auf der Kippe steht: bei wenigen Sampling-Schritten, bei hoher Guidance und bei Feinstrukturen. Was das im Einzelnen bedeutet, steht in [Sampling-Steps erklärt](#) und [CFG-Scale: Guidance in KI-Bildgeneratoren](#). Und wenn du dich fragst, warum ausgerechnet Hände so gern kippen: [Warum KI keine Hände kann](#) beantwortet das.

Was du dagegen tun kannst – und was nicht

ComfyUI kennt einen Schalter, der PyTorch auf langsamere, deterministische Algorithmen festlegt. Die Hilfe dazu ist erfrischend ehrlich: Das mache Bilder nicht in allen Fällen deterministisch. Das ist keine Nachlässigkeit, sondern eine korrekte Beschreibung der Lage. Deterministische Kernel garantieren, dass dieselbe Operation auf derselben Hardware dasselbe Ergebnis liefert. Sie garantieren nicht, dass eine Operation über einen Stapel von vier Bildern dasselbe liefert wie vier Operationen über je ein Bild – das sind schlicht verschiedene Rechnungen.

Die einzige Einstellung, die dir bit-genaue Wiederholbarkeit tatsächlich einbringt, ist batch_size 1. Damit fällt die Stapelgröße als Variable weg, und du bist nur noch von den übrigen Faktoren abhängig – die du im Griff haben kannst, weil sie sich protokollieren und einfrieren lassen.

Störquelle	Wirkung auf das Ergebnis	Gegenmittel
batch_size	andere Rechenreihenfolge, Abweichung bis hin zu anderem Bildaufbau	in der Produktion auf 1 setzen
Grafikkarte, Treiber, PyTorch-Version	andere Kernel, andere Rundung, teils sichtbare Unterschiede	Produktionsrechner als feste Umgebung führen, Versionen dokumentieren
Attention-Backend (etwa SDPA, Flash, Sage)	unterschiedliche Näherungen und Reduktionsreihenfolgen	Startparameter des Dienstes festschreiben, nicht ad hoc ändern
Quantisierung des Modells (FP8, GGUF)	geringere Genauigkeit, Ergebnis weicht von der vollen Gewichtsdatei ab	Variante der Gewichtsdatei mit Prüfsumme im Protokoll führen
Modell-, LoRA- oder VAE-	anderes Bild, auch bei identischem	Dateiname und Prüfsumme notieren,

Version	Seed	Dateien nicht überschreiben
Rauschquelle (CPU oder GPU)	anderes Startrauschen bei identischem Seed	Einstellung dokumentieren, nicht zwischen Rechnern mischen
experimentelle Beschleunigungsoptionen	ausdrücklich als qualitätsverändernd gekennzeichnet	in der Produktion abschalten

Ein Seed allein ist kein Nachweis

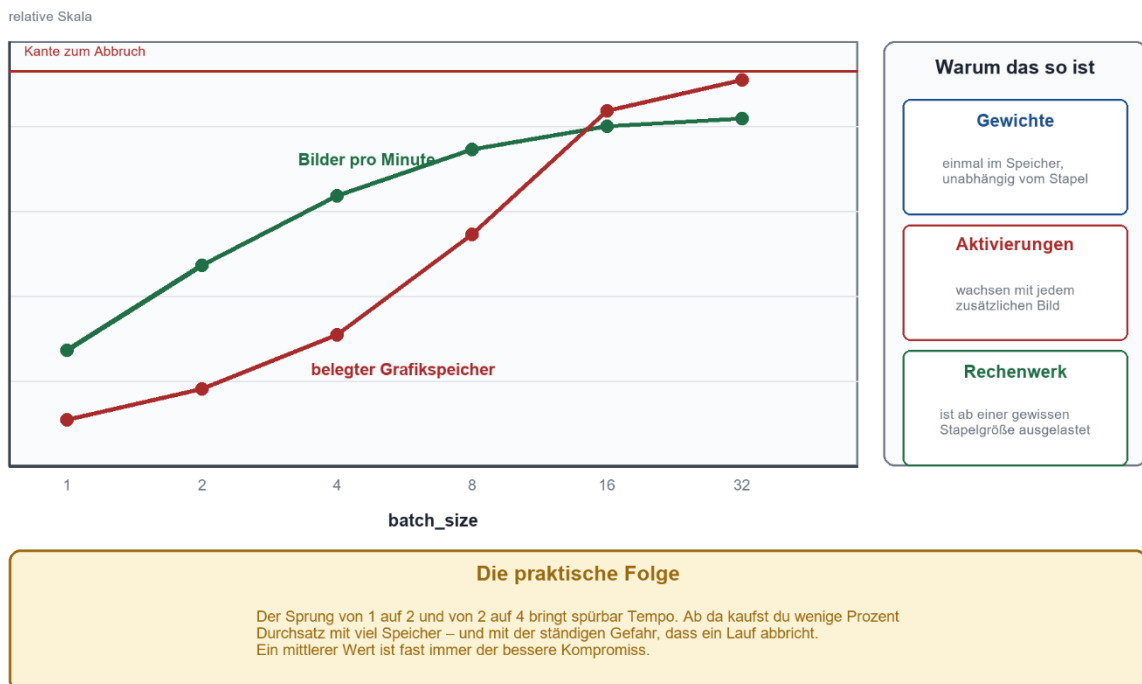
Wer in einer Dokumentation nur „Seed 1734921“ stehen hat, hat nichts dokumentiert. Die Zahl ist ein Zeiger in einen Zufallsstrom, und der Rest der Umgebung entscheidet, worauf sie zeigt. Nach einem Modell-Update, einem Treiberwechsel oder einer neuen ComfyUI-Version kann dieselbe Zahl ein anderes Bild bedeuten.

Wenn Reproduzierbarkeit Teil einer Freigabe oder eines Nachweises ist, gehört die komplette Umgebung ins Protokoll – und das Ergebnisbild selbst archiviert, nicht nur die Anleitung, es wiederherzustellen.

Speicher, Tempo und der Punkt, ab dem sich Stapeln nicht mehr lohnt

Bleibt die Frage, wie groß der Stapel in der Explorationsphase überhaupt sein sollte. Die Antwort hängt an zwei Kurven, die in unterschiedliche Richtungen laufen.

Der Ertrag großer Stapel läuft aus – der Speicherbedarf nicht



Skizze 3: Der Durchsatz läuft in eine Sättigung, der Speicherbedarf wächst weiter. Der sinnvolle Arbeitsbereich liegt dort, wo sich die beiden Kurven noch nicht verabschiedet haben.

Warum große Stapel irgendwann nichts mehr bringen

Der Tempogewinn eines Stapels entsteht dadurch, dass die Grafikkarte bei einem einzelnen Bild Leerlauf hat. Ist sie erst einmal ausgelastet, bringt jedes weitere Bild im Stapel keinen zusätzlichen Durchsatz mehr, sondern kostet nur noch Speicher und verlängert die Zeit bis zum ersten sichtbaren Ergebnis. Der Sprung von 1 auf 2 und von 2 auf 4 ist meist deutlich spürbar, danach

wird es schnell zäh. Wer auf 32 hochdreht, kauft ein paar Prozent Durchsatz mit einem erheblichen Abbruchrisiko.

Dazu kommt ein Faktor, der oft übersehen wird: Die Auflösung geht quadratisch in den Speicherbedarf der Aktivierungen ein. Eine batch_size, die bei 1024 mal 1024 problemlos läuft, kann bei 1536 mal 1536 bereits abbrechen. Wenn du also Auflösung und Stapelgröße gleichzeitig erhöhst, multiplizieren sich die Effekte.

Modellgröße verschiebt die Rechnung komplett

Die zweite Verschiebung kommt von den Modellen selbst. Bei einem SDXL-Modell ist der Anteil der Gewichte am Speicher überschaubar, da bleibt auf einer 24-GB-Karte reichlich Platz für einen Stapel. Bei der aktuellen FLUX-Generation sieht die Welt anders aus: FLUX.2 [dev] von Black Forest Labs, im November 2025 veröffentlicht, bringt 32 Milliarden Parameter mit. In voller Genauigkeit sind das rund 64 GB allein für die Gewichte, in FP8 immer noch etwa 32 GB, und selbst eine stark quantisierte GGUF-Variante liegt bei ungefähr 19 GB – auf einer 24-GB-Karte nur dann lauffähig, wenn der Textencoder in den Hauptspeicher ausgelagert wird. Dazu kommt, dass FLUX.2 als Textencoder ein multimodales Sprachmodell mit 24 Milliarden Parametern verwendet, das ebenfalls untergebracht sein will.

Auf solcher Hardware ist die Frage nach der optimalen batch_size müßig: Sie lautet 1, und die eigentliche Optimierungsaufgabe verlagert sich auf Quantisierung und Auslagerung. Welches Modell in welche Klasse fällt, steht in [Die 15 wichtigsten Modelle zur lokalen Bildgenerierung](#). Wie sich die Auflösung auf den Speicherbedarf auswirkt, behandelt [Auflösung & Seitenverhältnis bei KI-Bildern](#). Und wenn die FLUX-Generation bei dir in der Office-Welt landen soll, ist [Black Forest FLUX.2 in PowerPoint und Copilot](#) der passende Einstieg.

Situation	sinnvolle batch_size	Begründung
Ideensuche, kleines Modell, viel VRAM	4 bis 8	Durchsatz zählt, Einzelseeds sind in dieser Phase irrelevant
Ideensuche, wenig VRAM	1, dafür mehr Durchgänge	kein Abbruchrisiko, gleiche Bildmenge, nur später fertig
großes Modell der FLUX-Klasse	1	die Gewichte füllen den Speicher, für Stapel bleibt nichts übrig
hohe Auflösung ab 1536 Pixel Kantenlänge	1 bis 2	der Speicherbedarf der Aktivierungen steigt quadratisch mit der Kantenlänge
Vergleich einzelner Einstellungen	1	sonst überlagert die Stapelgröße den Effekt, den du messen willst
Produktion, Freigabe, Nachweispflicht	1	nur so ist der Lauf exakt wiederholbar
automatisierte Serie über eine Prompt-Liste	1 je Auftrag	jeder Auftrag bekommt eigenen Seed und eigenen Protokolleintrag

Die Stapelgröße einmal sauber austesten

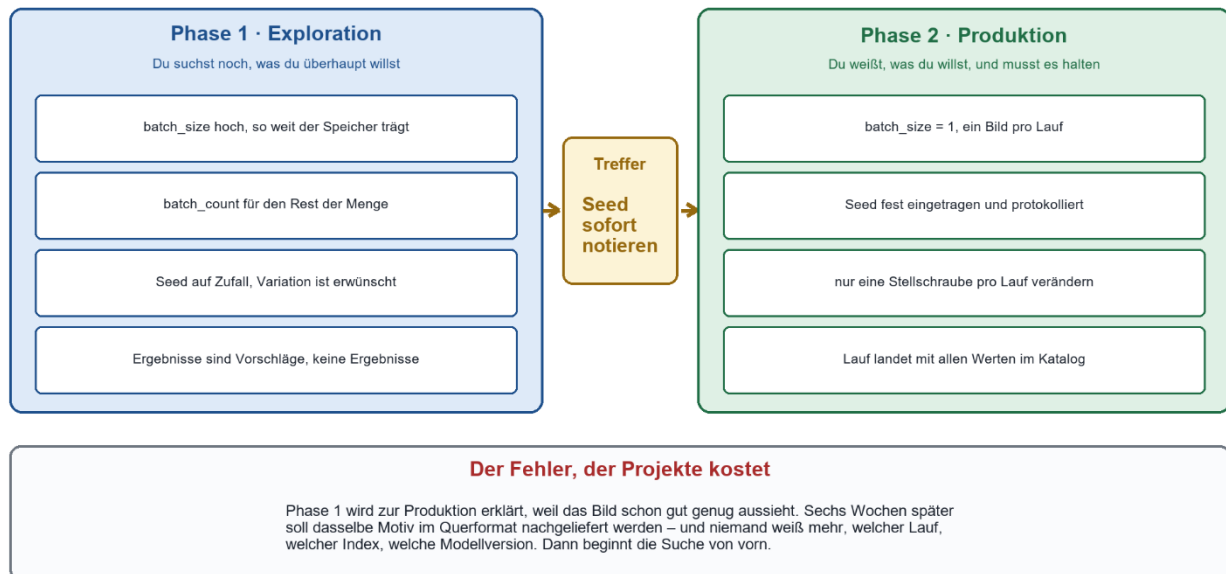
Taste dich einmal mit deinem Produktivmodell und deiner Zielauflösung an die Grenze heran: batch_size verdoppeln, bis es abbricht, dann eine Stufe zurück – und davon noch eine Stufe Sicherheitsabstand, weil ein Zusatzknoten oder ein zweiter Nutzer am selben Rechner die Rechnung sonst kippt. Diesen Wert notierst du als Obergrenze für die Explorationsphase. Danach musst du nie wieder raten.

Miss dabei nicht nur, ob es läuft, sondern auch die Bilder pro Minute. In den meisten Aufbauten liegt das Optimum deutlich unter dem, was gerade noch in den Speicher passt.

Arbeiten wie in der Produktion: Einzelbild, fester Seed, Protokoll

Die saubere Arbeitsteilung ist kein Dogma, sondern eine Übergabe zwischen zwei Phasen mit unterschiedlichen Zielen. In der einen suchst du, in der anderen sicherst du. Wer beides mit denselben Einstellungen macht, verliert entweder Zeit oder Nachvollziehbarkeit.

Zwei Phasen, zwei Einstellungen – die Übergabe ist der Seed



Skizze 4: Der Übergang ist der kritische Moment. Wer den Seed dort nicht festhält, verlagert die Arbeit nur nach hinten.

Phase 1: stöbern, ohne sich etwas zu merken

In der Explorationsphase weißt du noch nicht, was du willst. Hier ist ein großer Stapel genau richtig: Seed auf Zufall, batch_size bis an die getestete Obergrenze, batch_count für die Menge. Die Ergebnisse sind Vorschläge, keine Ergebnisse. Du musst dir in dieser Phase bewusst nichts merken, und das ist der eigentliche Wert: Der Kopf bleibt frei für die Frage, ob die Bildidee überhaupt trägt.

Wenn in dieser Phase systematisch gar nichts Brauchbares herauskommt, ist die Stapelgröße nicht das Problem. Dann liegt es am Prompt, am Modell oder an der Guidance – siehe [Prompt und Negativ-Prompt in der KI-Bildgenerierung](#) und [Warum die KI den Prompt ignoriert](#). Mehr Würfel zu werfen macht die Würfel nicht besser.

Phase 2: festnageln, bevor du weiterarbeitest

Sobald ein Kandidat überzeugt, wechselst du die Betriebsart. Ab hier gilt: batch_size 1, Seed fest eingetragen, und pro Lauf wird genau eine Stellschraube verändert. Das ist nicht nur ordentlich, sondern schlicht die einzige Möglichkeit, den Effekt einer Änderung überhaupt zu sehen. Wer Sampler, Schritte und Guidance gleichzeitig anfasst und danach sagt, das Bild sei besser geworden, hat keine Erkenntnis gewonnen, sondern nur ein anderes Bild.

Diese Disziplin ist der Unterschied zwischen einer Person, die schöne Bilder erzeugt, und einem Team, das verlässlich liefert. Genau das ist der Inhalt von [KI-Bildgenerierung Schulung](#): nicht, welcher Regler wie heißt, sondern wie man in kontrollierten Schritten arbeitet und das Ergebnis so dokumentiert, dass es der nächste Kollege ein halbes Jahr später noch nachvollziehen kann.

Was tatsächlich ins Protokoll gehört

Beide großen Werkzeuge legen Metadaten in der PNG-Datei ab – die WebUI-Familie einen Parameterstring mit dem Einzelseed, ComfyUI den kompletten Arbeitsablauf als strukturierte Daten. Das ist ausgesprochen nützlich und trotzdem keine Dokumentation. Der Grund ist banal: Sobald ein Bild als JPEG oder WebP exportiert, durch ein Bildbearbeitungsprogramm geschickt oder in ein Content-Management-System geladen wird, sind diese Metadaten in aller Regel weg. Was du brauchst, ist ein Eintrag außerhalb des Bildes.

Feld	Beispiel	Warum es gebraucht wird
Seed	1734921	der Zeiger in den Zufallsstrom
batch_size des Laufs	1	ohne diese Angabe ist der Seed mehrdeutig
Index im Stapel	0	nur nötig, wenn doch gestapelt wurde – dann aber zwingend
Modelldatei mit Variante	flux2-dev, FP8-Variante	Quantisierung und Version verändern das Ergebnis
LoRA und Gewichtung	marke-produktfoto, 0.65	ein anderer Wert liefert ein anderes Bild
Sampler und Scheduler	Name beider Komponenten	beides zusammen bestimmt den Rechenweg
Schritte und Guidance	28 Schritte, Guidance 3.5	die zwei Werte, an denen am häufigsten gedreht wird
Auflösung	1024 × 1024	verändert Bildaufbau und Speicherbedarf
Prompt und Negativ-Prompt	vollständig, nicht sinngemäß	ein umgestelltes Wort reicht für ein anderes Bild
Umgebung	ComfyUI-Version, Grafikkarte, Treiber	die stillen Variablen, die am häufigsten zuschlagen

Handgeführt hält das keine Woche durch. In meinen eigenen Aufbauten läuft das über zwei Werkzeuge: UB.Forge erzeugt die Bilder, schreibt jeden Lauf mit allen Werten mit und prüft vor dem Start, ob die Kombination aus Modell, Auflösung und Stapelgröße überhaupt plausibel ist – die Meldung „out of memory“ kommt dann vor dem Lauf und nicht mittendrin. UB.Dossier nimmt das Ergebnis auf, führt den Katalog und hält den Nachweis, welches Bild aus welchem Lauf stammt. Wie so ein Katalog auf Microsoft-Technik aussehen kann, beschreibt [Digital Asset Management für KI-Bilder mit SharePoint Online und Power Platform](#); die Bildproduktion dahinter ist Thema von [Aufbau einer skalierbaren Bild-Maschine mit ComfyUI und Flux](#).

Der Nachweis ist inzwischen nicht mehr nur eine Frage der Ordnungsliebe. Seit dem 2. August 2026 greifen die Transparenzpflichten des EU AI Act für synthetisch erzeugte Inhalte; was das konkret bedeutet, steht in [EU AI Act: Was seit dem 2. August 2026 gilt](#). Ein Lauf, den du nicht rekonstruieren kannst, ist auch ein Lauf, über den du keine Auskunft geben kannst. Wer Bilder für die Marke erzeugt, hat ohnehin ein zweites Interesse daran, dass ein Motiv exakt wiederholbar bleibt – dazu [Markenkonsistente KI-generierte Bilder](#).

Die Fünf-Sekunden-Regel für den Treffer

In dem Moment, in dem dir ein Bild aus dem Stapel gefällt, machst du genau eine Sache, bevor du irgendetwas anderes tust: Du schreibst Seed, batch_size und Index auf. Nicht später, nicht nachdem du noch schnell eine Variante probiert hast – denn dann hat die Oberfläche den Seed womöglich schon weitergedreht.

In ComfyUI hilft dabei die Einstellung neben dem Seed-Feld, die nach jedem Lauf entscheidet, ob der Wert fest bleibt, hoch- oder heruntergezählt oder neu gewürfelt wird. Für die Produktionsphase steht sie auf „fest“. Wer sie auf „zufällig“ stehen lässt und erst nach dem nächsten Lauf nachsieht, findet den alten Wert nicht mehr im Feld.

Was batch_size nicht ist

Zum Schluss dieses Kapitels die Klarstellung, die sich hartnäckig hält: batch_size ist kein Qualitätsregler. Ein größerer Stapel liefert mehr Bilder zur Auswahl, aber kein besseres einzelnes Bild. Die Trefferwahrscheinlichkeit steigt, weil du mehr Würfe machst – nicht, weil die Würfe besser würden. Die Bildqualität hängt an Prompt, Modell, Sampler, Schritten und Guidance, also an [Sampler & Scheduler erklärt](#), [Sampling-Steps erklärt](#) und [Checkpoints, Modelle und VAE](#). Der Stapel ist eine Transportfrage, keine Qualitätsfrage.

Häufige Fragen

Was ist der Unterschied zwischen batch_size und batch_count?

batch_size rechnet mehrere Bilder gleichzeitig in einem einzigen Durchgang durch das Modell – schneller, aber mit Speicherbedarf, der mit jedem zusätzlichen Bild steigt. batch_count rechnet die Bilder nacheinander in getrennten Durchgängen – langsamer, aber ohne zusätzlichen Speicherbedarf, weil nach jedem Durchgang wieder aufgeräumt wird. Die Gesamtzahl der Bilder ist batch_size mal batch_count.

Beeinflusst batch_size die Bildqualität?

Nicht in dem Sinn, dass größere Stapel bessere Bilder machen würden. batch_size ist ein Mengen- und Tempo-Regler. Was allerdings stimmt: Die Stapelgröße kann das konkrete Ergebnis verändern, weil sie die Reihenfolge der Rechenoperationen beeinflusst. Das ist kein Qualitätsunterschied, sondern ein Reproduzierbarkeitsproblem – und es ist der Grund, warum in der Produktion batch_size 1 gilt.

Bekomme ich im Stapel exakt dasselbe Bild wie bei einem Einzellauf?

Nicht garantiert. Bei gleichem Seed startet das Bild an Index 0 in ComfyUI zwar mit demselben Rauschen wie ein Einzellauf, aber der Rechenweg durch das Modell ist bei einem Stapel ein anderer. Fließkomma-Abweichungen in den letzten Stellen verstärken sich über die Sampling-Schritte. Meist siehst du keinen Unterschied, manchmal schon. Wer bit-genaue Wiederholbarkeit braucht, rechnet einzeln.

Haben alle Bilder in einem Stapel denselben Seed?

Das hängt am Werkzeug. In der Automatic1111-Familie und ihren Ablegern bekommt jedes Bild eine eigene, fortlaufend hochgezählte Seed-Nummer, die auch in den PNG-Metadaten landet. In ComfyUI gilt ein einziger Seed für den gesamten Stapel; die Bilder unterscheiden sich, weil jedes einen anderen Abschnitt desselben Rauschstroms bekommt. Eigene Seed-Nummern haben sie dort nicht.

Wie hole ich ein bestimmtes Bild aus einem ComfyUI-Stapel zurück?

Du brauchst drei Angaben: den Seed des Laufs, die batch_size des Laufs und die Indexposition des Bildes. Damit stellst du entweder denselben Lauf noch einmal nach und greifst dasselbe Bild an derselben Position ab – oder du setzt den Knoten „Latent aus Batch“ ein, trägst den Index ein und rechnest nur dieses eine Bild. Was nicht funktioniert: den Seed um die Indexposition zu erhöhen. Das liefert etwas völlig anderes.

Warum bekomme ich eine „out of memory“-Meldung?

Weil die Summe aus Modellgewichten, Textencoder und den Zwischenergebnissen aller parallel gerechneten Bilder nicht in den Grafikspeicher passt. Die schnellste Abhilfe ist, batch_size zu senken und die Menge stattdessen über mehr Durchgänge zu holen. Häufig übersehen: Auch die Auflösung geht in diese Rechnung ein, und zwar quadratisch. Wenn das Senken der Stapelgröße nicht reicht, ist entweder das Modell zu groß für die Karte oder die Auflösung zu hoch.

Soll ich große Stapel oder Einzelbilder erzeugen?

Beides, nur nicht gleichzeitig. In der Suchphase große Stapel, weil dich die einzelnen Seeds nicht interessieren und Durchsatz zählt. Sobald ein Kandidat überzeugt, wechselst du auf batch_size 1 mit festem, notiertem Seed und arbeitest von dort kontrolliert weiter. Der Fehler ist nicht das Stapeln, sondern der fehlende Wechsel.

Was tue ich, wenn mir ein Bild aus einem großen Stapel gefällt?

Sofort Seed, batch_size und Indexposition notieren, bevor du irgendetwas anderes machst. In der WebUI-Familie kannst du den Einzelseed zur Not noch aus den PNG-Metadaten auslesen, solange die Datei unverändert ist. In ComfyUI existiert diese Nummer gar nicht – dort ist die Position im Stapel die Information, und die geht verloren, sobald die Dateien umbenannt oder umsortiert werden.

Welche batch_size ist bei großen Modellen wie FLUX.2 sinnvoll?

In aller Regel 1. FLUX.2 [dev] bringt 32 Milliarden Parameter mit und belegt selbst in reduzierter Genauigkeit den größten Teil einer Consumer-Grafikkarte; dazu kommt ein Textencoder in der Größenordnung eines eigenständigen Sprachmodells. Für parallele Bilder bleibt da schlicht nichts übrig. Die Optimierung findet auf dieser Modellgeneration bei Quantisierung und Auslagerung statt, nicht bei der Stapelgröße.

Reicht es, wenn der Seed in den Bildmetadaten steht?

Für die eigene Arbeit am selben Tag ja. Für alles andere nein. Metadaten überleben keinen Export nach JPEG, keine Bildbearbeitung und in der Regel auch keinen Upload in ein Content-Management-System. Und selbst wenn sie erhalten bleiben: Der Seed allein sagt nichts über Modellversion, Quantisierung, Software-Stand und Hardware. Ein Protokoll außerhalb des Bildes ist kein Bürokratie-Zuschlag, sondern die einzige Form, die einen Wiederherstellungsversuch überlebt.

Kann ich mit batch_size Varianten eines Bildes erzeugen?

Nur zufällige, keine kontrollierten. Ein Stapel liefert vier unabhängige Bilder aus vier verschiedenen Rauschabschnitten – sie haben denselben Prompt gemeinsam, sonst nichts. Für gezielte Varianten eines konkreten Bildes brauchst du andere Werkzeuge: einen Variations-Seed mit kleiner Variationsstärke, eine Bild-zu-Bild-Verarbeitung mit niedriger Stärke oder gezielte Änderungen am Prompt. Der Unterschied ist der zwischen „noch ein Bild“ und „dasselbe Bild, leicht anders“.

Fazit

batch_size ist der harmloseste Regler der ganzen Oberfläche und der, der die meisten Projekte im Nachhinein Geld kostet. Nicht weil er etwas kaputt macht, sondern weil er die Illusion erzeugt, man käme schneller voran. Man kommt tatsächlich schneller voran – so lange, bis jemand ein bestimmtes Bild zurückhaben will.

Die Merksätze sind kurz. batch_size rechnet nebeneinander und kostet Speicher, batch_count rechnet hintereinander und kostet Zeit. Im Stapel vergeben die Werkzeuge den Seed unterschiedlich: Die WebUI-Familie zählt hoch, ComfyUI zieht weiter aus demselben Strom und kennt nur Indizes. Und die Stapelgröße selbst ist eine Variable im Ergebnis, keine neutrale Einstellung – wer exakt dasselbe Bild wiederhaben muss, rechnet einzeln.

Die praktische Konsequenz ist eine Frage der Betriebsdisziplin, nicht der Technik: zwei Phasen, ein klarer Übergabepunkt, und an diesem Punkt ein Protokoll, das mehr enthält als eine Zahl. Wer das einmal eingerichtet hat, spart sich die Suche nach dem Bild, das es angeblich mal gab. Wenn du das

für ein Team aufsetzen willst – von der Hardware über den Arbeitsablauf bis zum Nachweis – ist [KI-Bildgenerierung Beratung](#) der passende Einstieg; die strategische Einordnung dahinter steht in [KI-Bildgenerierung im Unternehmen – Die strategische Alternative zur Cloud](#).

Weiterlesen

- › Alles Übrige zum Thema im Überblick: [KI-Bildgenerierung](#)
- › Die Zahl, um die es eigentlich geht: [Der Seed bei der KI-Bildgenerierung](#)
- › Woran die Qualität wirklich hängt: [Sampler & Scheduler erklärt](#) und [Sampling-Steps erklärt](#)
- › Wenn der Speicher der Engpass ist: [Die 15 wichtigsten Modelle zur lokalen Bildgenerierung und Cloud vs. lokale KI-Bildgenerierung](#)
- › Vom Einzelbild zur Produktion: [Aufbau einer skalierbaren Bild-Maschine mit ComfyUI und Flux](#)
- › Woran man erzeugte Bilder überhaupt erkennt: [KI-generierte Bilder erkennen](#)
- › Warum die Bilder hier selbst erzeugt sind: [Dogfooding: Ich setze ein, was ich berate](#)