LM Studio Bionic – Die komplette Anleitung

Stand: Juli 2026

Inhaltsverzeichnis

> **Hinweis vorab:** Bionic ist zum Zeitpunkt dieser Anleitung erst wenige Tage alt (offizieller Launch Mitte Juli 2026). Manche Details – insbesondere die konkreten Preise des „Bionic Pass”-Plans – sind vom Hersteller selbst noch als „coming soon” markiert und können sich kurzfristig ändern. Wo das der Fall ist, wird es unten explizit vermerkt.

1. Was ist LM Studio Bionic?

LM Studio Bionic ist eine eigenständige KI-Agenten-App für offene (Open-Weight-)Modelle, entwickelt vom Team hinter LM Studio. Bionic nutzt die etablierte LM-Studio-Laufzeitumgebung, ist aber technisch eine **zweite, separate Anwendung**: LM Studio (die klassische App) bietet eine Chat-Oberfläche, detaillierte Modellverwaltung und lokale APIs. Bionic ergänzt diese Produktfamilie um agentisches Arbeiten: Es kann in einem lokalen Projektordner Dateien lesen und ändern, Shell-Befehle ausführen, mit Git arbeiten, Dokumente verarbeiten und neuerdings auch lokal per Sprache gesteuert werden.

Kurz gesagt: Die klassische LM-Studio-App richtet sich stärker an Chat, Modellkonfiguration und Entwickler-APIs; Bionic ist der eigenständige Agent für Coding, Recherche und Dokumentenarbeit. Die klassische App muss nicht installiert sein, damit Bionic lokale Modelle ausführen kann.

Homepage: https://lmstudio.ai/

2. Wem gehört LM Studio Bionic?

LM Studio und Bionic werden von Element Labs, Inc. entwickelt. Die lokale Basisnutzung wird kostenlos angeboten; zusätzlich gibt es kostenpflichtige Cloud-Inferenz (siehe Kapitel 9, „Preise"). Datenschutz und lokale Ausführung stehen im Zentrum der Produktbeschreibung, mit „Zero Data Retention" als Versprechen für die Cloud-Funktionen von Bionic.

3. Architektur: Wie hängen LM Studio und Bionic zusammen?

- **LM Studio (Basis-App):** Modell-Download-Manager, Chat-Oberfläche, lokaler OpenAI-kompatibler Server, Presets (System Prompts + Parameter). Läuft mit `llama.cpp` (GGUF-Modelle) oder mit Apples `MLX`-Framework auf Apple-Silicon-Macs.
- **LM Studio Bionic (Agenten-App):** Eigenständiges Programm mit eigenem Modell-Download-Manager (Settings > Local Models > Explore/Library, siehe Kapitel 5 und 7.1) – lokale Modelle lassen sich komplett innerhalb von Bionic herunterladen, laden und verwalten, ohne die klassische App dafür starten zu müssen. Alternativ lassen sich auch Frontier-Modelle in der „LM Studio Secure Cloud" ansprechen. Bionic bringt zusätzlich Projekt-Verwaltung, Datei-/Git-/Shell-Werkzeuge, einen Bereich für Connected Apps, Sprachsteuerung und automatische Checkpoints mit.
- Bionic verwendet die LM-Studio-Runtime (`llama.cpp`/`MLX`). Dass beide Apps in jeder Version automatisch denselben Modellbestand und sämtliche Konfigurationen gemeinsam verwalten, ist in der aktuellen Bionic-Dokumentation jedoch nicht eindeutig zugesichert. Für **Presets** (gespeicherte System-Prompt-/Parameter-Bündel, Kapitel 7.2) und besonders feingranulare technische Einstellungen kann die klassische LM-Studio-App zusätzlich sinnvoll sein. Eine automatische Übernahme klassischer Presets in Bionic ist derzeit nicht dokumentiert.

4. Installation

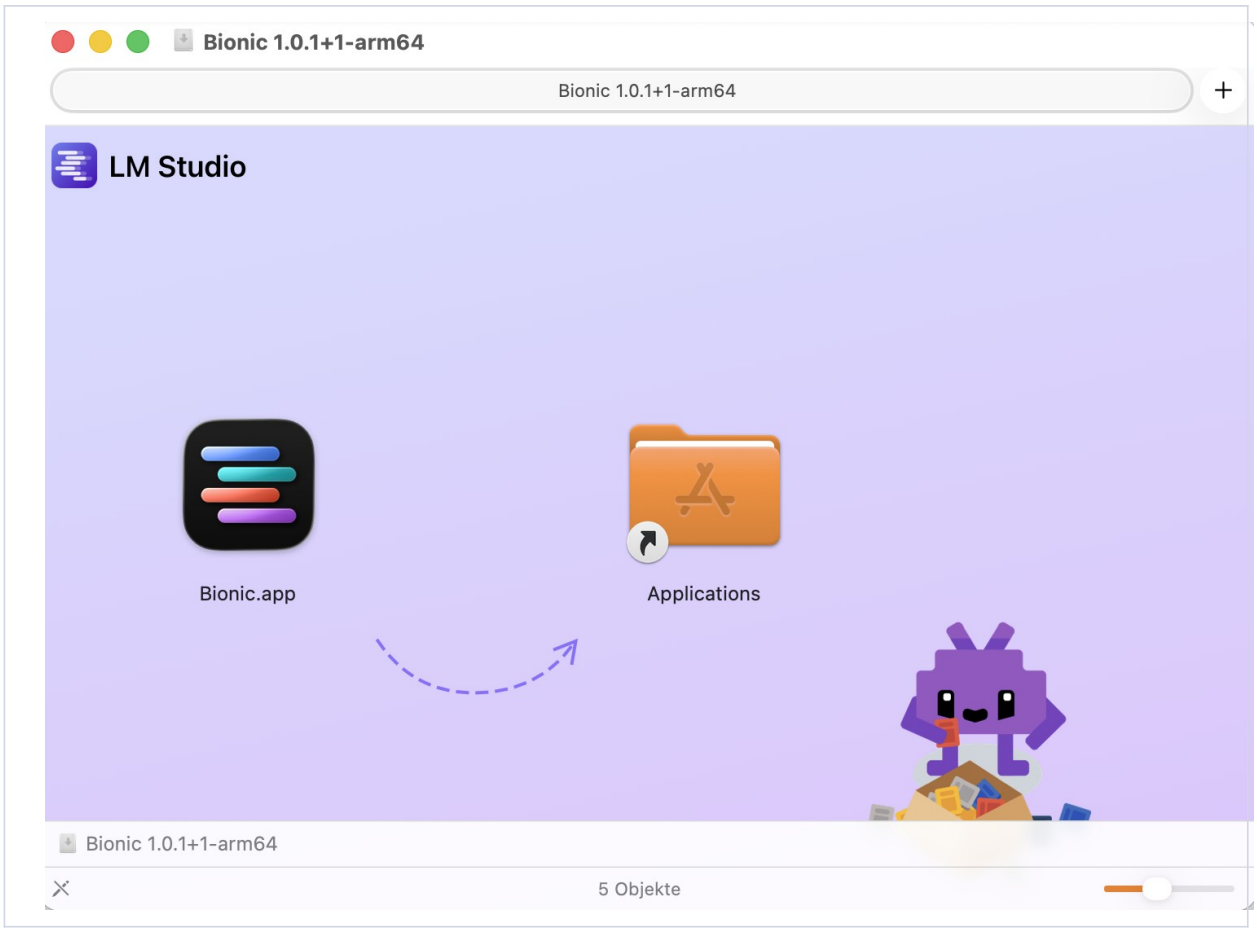
4.1 Systemanforderungen (macOS)

- **Chip:** Apple Silicon (M1/M2/M3/M4) – Intel-Macs werden nicht unterstützt
- **Betriebssystem:** Für die klassische LM-Studio-App gilt mindestens macOS 13.4; MLX-Modelle setzen macOS 14 oder neuer voraus. Für Bionic sollte zusätzlich die aktuelle Angabe auf der Downloadseite geprüft werden, da die Bionic-Dokumentation noch keine eigene ausführliche Anforderungstabelle enthält.
- **Arbeitsspeicher:** 16 GB RAM empfohlen als praktische Untergrenze; 24 GB+ für ein wirklich angenehmes Arbeiten mit etwas größeren lokalen Modellen; mit 8 GB laufen bestenfalls sehr kleine, stark quantisierte Modelle mit reduziertem Kontext

Diese Anleitung konzentriert sich auf macOS. Welche Bionic-Builds für Windows oder Linux angeboten werden, sollte auf der aktuellen Bionic-Downloadseite geprüft werden; die allgemeine LM-Studio-Systemanforderungsseite bezieht sich nicht automatisch auf jede Bionic-Preview.

4.2 Schritt-für-Schritt-Installation

1. Downloadseite öffnen: https://lmstudio.ai/
2. Installer für macOS (Apple Silicon) herunterladen und wie eine normale Mac-App installieren (in den Programme-Ordner ziehen)
3. Optional die klassische LM-Studio-App zusätzlich installieren, wenn Presets, detaillierte Modellkonfigurationen oder lokale Entwickler-APIs benötigt werden. Für lokale Modelle in Bionic ist dieser Schritt nicht erforderlich. Klassische Presets werden in LM Studio verwaltet; ihre Verwendung innerhalb von Bionic ist derzeit nicht dokumentiert (siehe Kapitel 7.2).
4. Bionic separat herunterladen und installieren (verlinkt auf derselben Seite bzw. unter https://lmstudio.ai/docs/bionic) – Bionic erscheint danach als eigenes Programm neben LM Studio, nicht als Tab innerhalb von LM Studio
5. Nur bei Bedarf ein LM-Studio-Konto anlegen. Cloud-Modelle benötigen ein angemeldetes Konto mit eingerichteter Abrechnung. Lokale Modelle und laut Dokumentation auch LM-Link-Modelle können ohne LM-Studio-Konto verwendet werden. Für die begrenzte ZDR-Websuche ist eine Anmeldung erforderlich.



Installation von LM Studio und Bionic

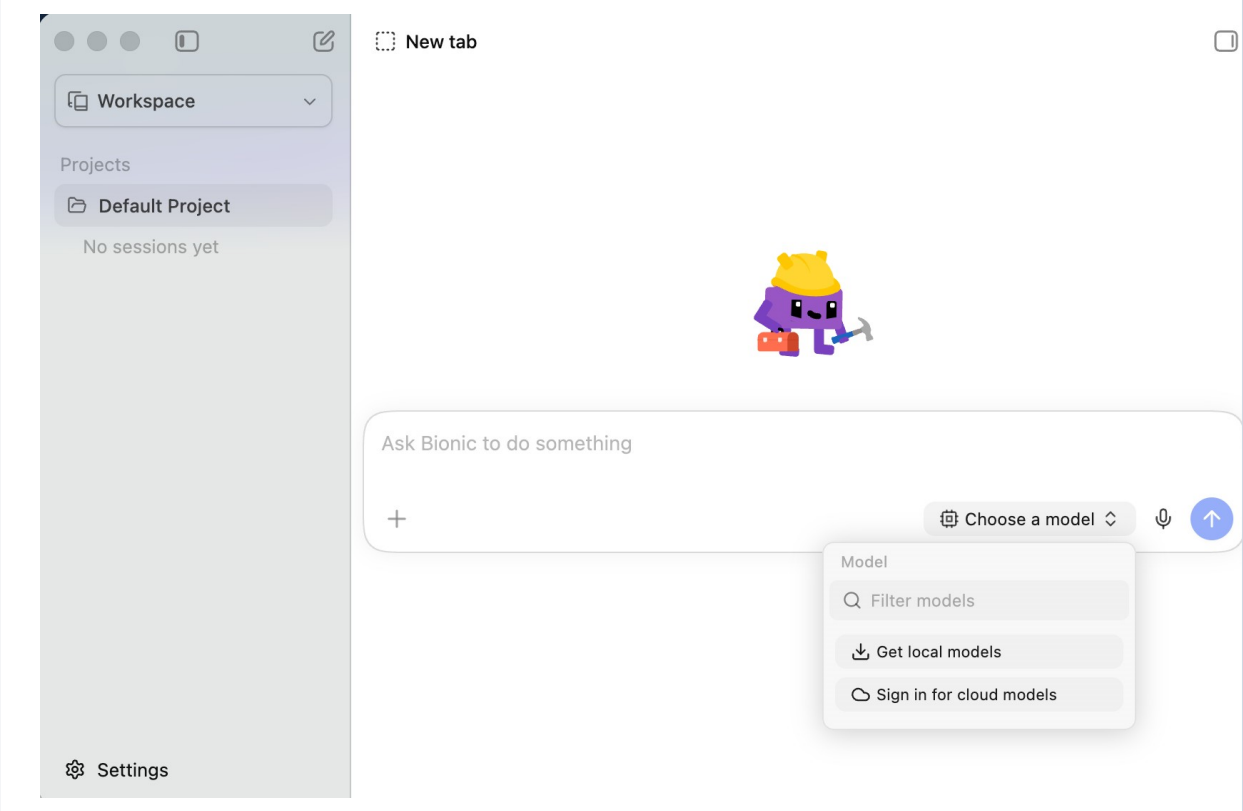
4.3 Aktualisieren

Für die klassische LM-Studio-App sind automatische Update-Prüfungen unter macOS und Windows dokumentiert. Bei Bionic sollte regelmäßig im App-Menü beziehungsweise auf der offiziellen Website nach Aktualisierungen gesucht werden. In einer frühen Preview können sich Oberfläche und Funktionen kurzfristig ändern.

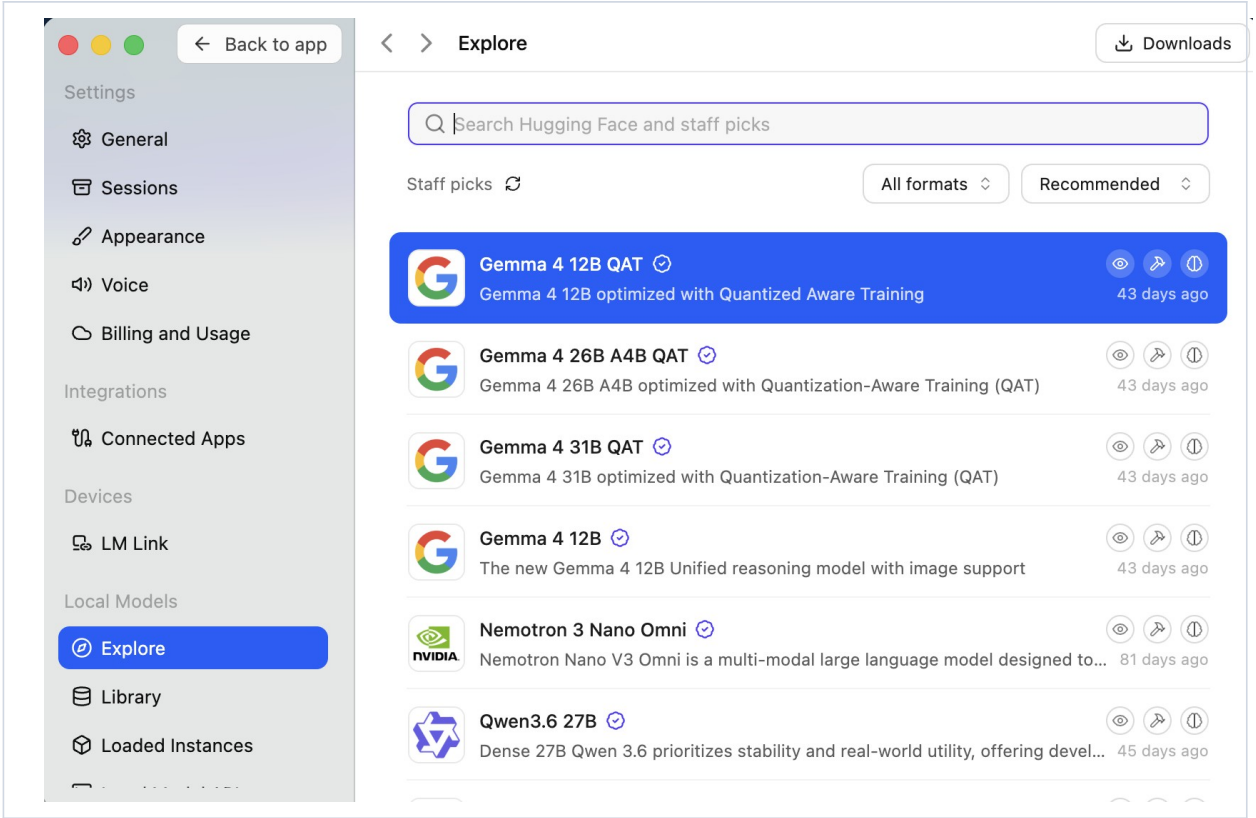
5. Ein lokales LLM laden

Ein lokales Modell lässt sich vollständig innerhalb von Bionic herunterladen und laden. Die klassische LM-Studio-App wird dafür nicht benötigt. Bionic verwendet dabei die LM-Studio-Runtime (siehe Kapitel 3). Wer die klassische App ohnehin installiert hat, findet dort ebenfalls einen Such- und Download-Bereich; die genaue Oberfläche unterscheidet sich jedoch von Bionic.

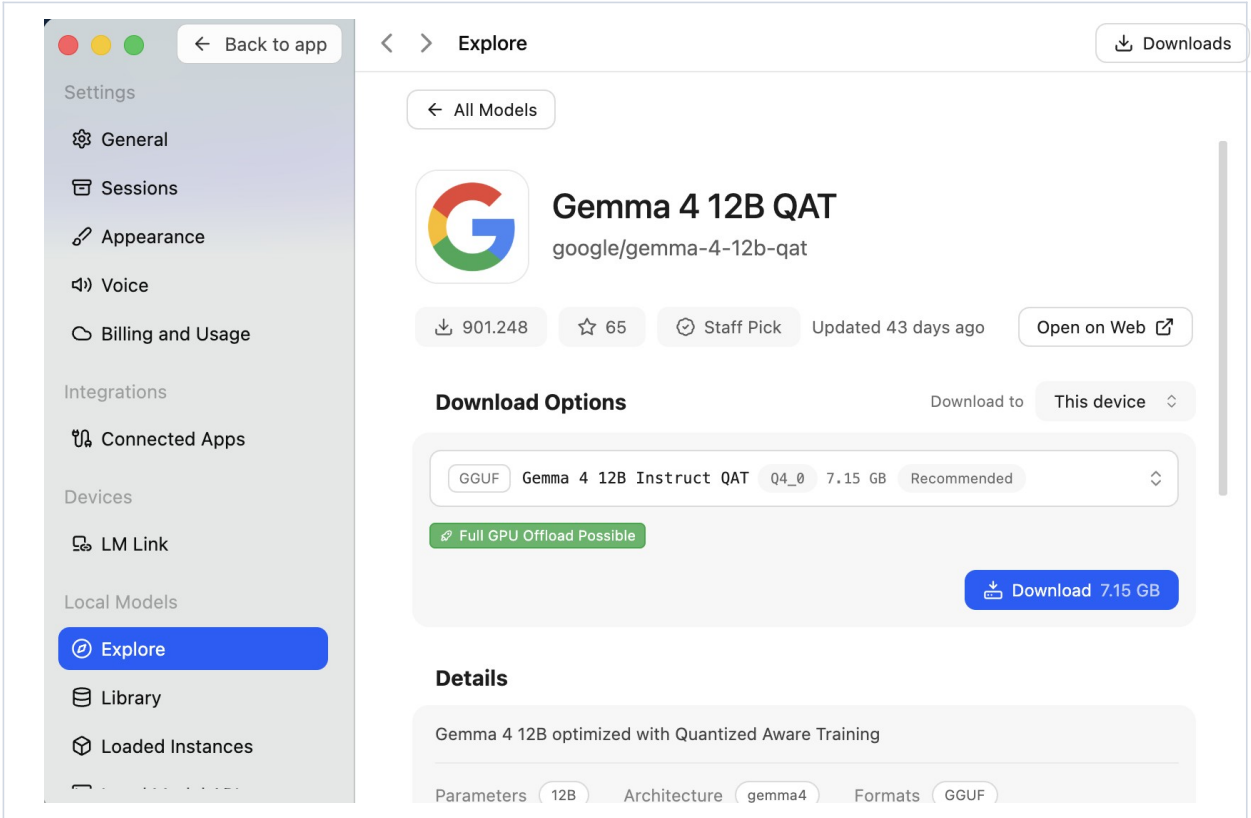
1. In Bionic **Settings** öffnen, unter **Local Models** den Reiter **Explore** wählen
2. Nach einem Modellnamen suchen (z. B. „qwen", „gemma", „llama") oder direkt eine Hugging-Face-URL einfügen
3. Aus den angebotenen Varianten eine Quantisierungsstufe wählen – der Buchstabe „Q" plus Zahl im Dateinamen (z. B. `Q4_K_M`, `Q8_0`) zeigt den Kompressionsgrad: niedrigere Zahl = kleinere Datei und weniger RAM-Bedarf, aber etwas geringere Qualität. Gängige Faustregel in der Community: „mindestens 4-Bit, wenn die Maschine es zulässt" – Q4_K_M gilt dabei meist als guter Standardkompromiss
4. Download starten – die Datei landet standardmäßig in einem LM-Studio-eigenen Modellverzeichnis, sichtbar danach unter **Local Models > Library**
5. Auf Apple-Silicon-Macs stehen viele Modelle zusätzlich im **MLX-Format** bereit. MLX ist für Apple-Chips optimiert; GGUF wird über llama.cpp ausgeführt und ist plattformübergreifend verbreitet. Welche Variante schneller ist, hängt von Modell, Quantisierung, Kontextlänge und Aufgabe ab. Ohne einen reproduzierbaren Benchmark sollte keine pauschale Prozentangabe genannt werden. Wo verfügbar, lohnt sich ein praktischer Vergleich auf dem eigenen Mac.
6. Unter **Local Models > Loaded Instances** lässt sich einsehen, welches Modell aktuell im Speicher liegt und wie viel RAM/VRAM es belegt (in der klassischen App per Tastenkombination `⌘ Shift R`)



Modellauswahl beim Download



Explore-Bereich mit LLM-Suche



Explore-Bereich mit Gemma-4-Modell

5.1 Modellempfehlungen nach Arbeitsspeicher (Richtwerte)

RAM	Geeignete Modellgröße	Beispiele
16 GB	kleine Modelle, einfache und klar begrenzte Aufgaben	z. B. Qwen 3.5 4B oder eine kleine quantisierte Gemma-4-Variante
24–32 GB	kleine bis mittlere Modelle; größere Varianten nur abhängig von Quantisierung und Kontext	Modellwahl über Bionics „Device Fit" prüfen
Viel RAM / dedizierte GPU / Cloud	anspruchsvolle Coding-Aufgaben	z. B. GLM 5.2 oder Kimi Code K2.7 in der Secure Cloud

Diese Werte sind nur grobe Orientierung. Der tatsächliche Bedarf hängt stark von Quantisierung, Kontextfenster, KV-Cache und weiteren gleichzeitig laufenden Programmen ab. Verlässlicher als eine starre RAM-Tabelle ist der „Device Fit"-Hinweis in Bionic.

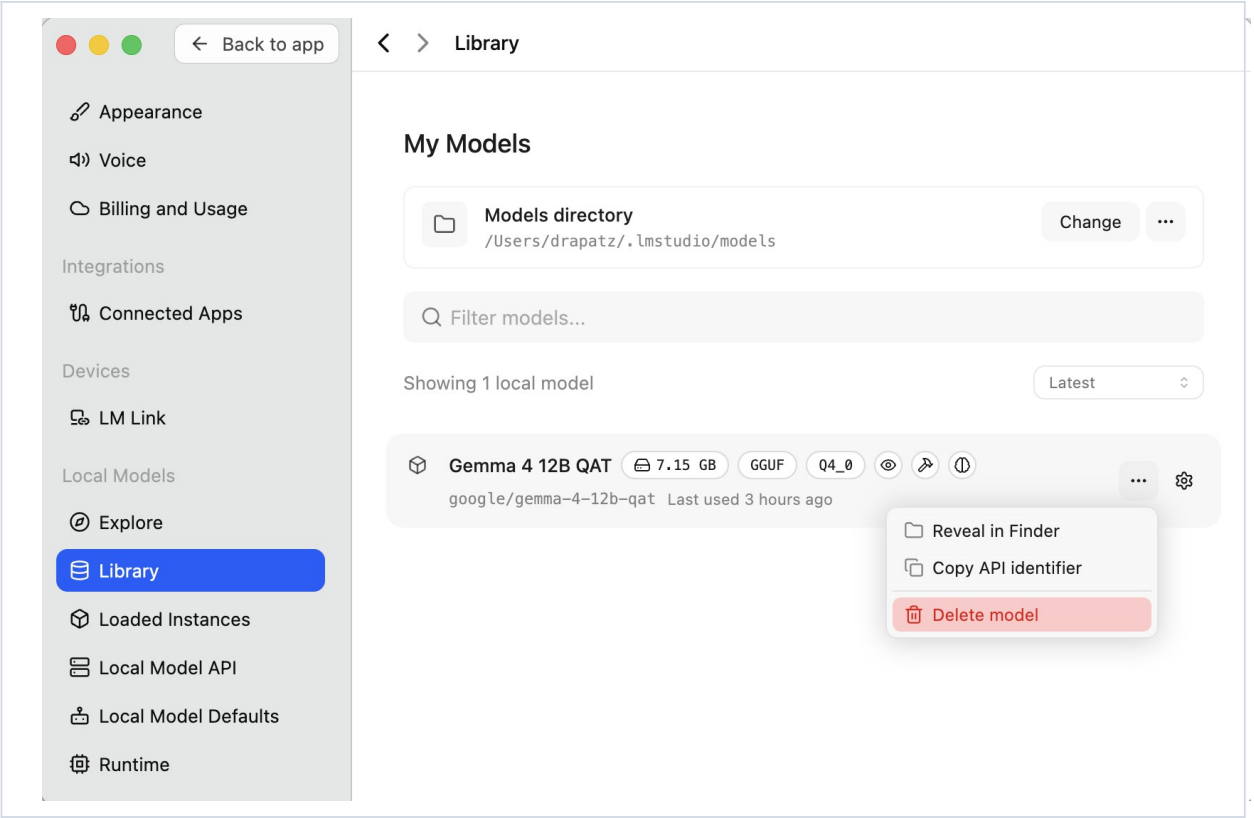
5.2 Modelle wieder löschen

Heruntergeladene Modelle können mehrere Gigabyte groß werden. Es lohnt sich also, nicht mehr benötigte Modelle gezielt wieder zu entfernen. Das geht direkt in Bionic, ohne die klassische App zu öffnen:

1. In Bionic **Settings** öffnen, unter **Local Models** den Reiter **Library** wählen
2. Beim gewünschten Modell das Drei-Punkte-Menü öffnen und **„Delete"** wählen
3. Das Modell darf dabei nicht gerade aktiv geladen sein – ist es das doch, vorher über **Local Models > Loaded Instances** entladen, sonst kann es zu einer Fehlermeldung kommen

In der klassischen App läuft derselbe Schritt über den Reiter **„My Models"** in der Seitenleiste (Drei-Punkte-Menü → „Delete"; ist das Modell aktiv geladen, vorher über `⌘ Shift R` entladen).

> **Historisch gemeldete Einschränkung:** In älteren Versionen wurde berichtet, dass nach dem Löschen über die Oberfläche ein zugehöriger Ordner auf der Festplatte zurückblieb. Das ist kein Beleg dafür, dass der Fehler in der aktuellen Bionic-Version weiterhin besteht. Gibt der Löschvorgang keinen sichtbaren Speicherplatz frei, sollte zunächst die aktuelle Library beziehungsweise der konfigurierte Modellordner geprüft werden.



Modell über das Drei-Punkte-Menü löschen

Eine GGUF-Modelldatei enthält Modelldaten, kann aber Teil einer mehrteiligen Ablage sein oder über Links von mehreren Werkzeugen genutzt werden. Manuelles Löschen ist daher nur ratsam, wenn Pfad und Zuordnung eindeutig geprüft wurden und das Modell nicht geladen ist. `lms unload` entlädt ein Modell lediglich aus dem Arbeitsspeicher und löscht es nicht von der Festplatte. Die aktuelle CLI-Referenz sollte vor einer Automatisierung auf einen inzwischen ergänzten Löschbefehl geprüft werden.

5.3 Der lokale Server (OpenAI-kompatibel)

LM Studio kann ein geladenes Modell zusätzlich über einen lokalen Server ansprechbar machen, den auch andere Programme nutzen können (z. B. eigene Skripte oder Drittanbieter-Tools):

- **OpenAI Compatibility API** – für Tools, die ohnehin die OpenAI-API-Struktur erwarten
- **LM Studio REST API v1** – die herstellereigene API für lokale Inferenz und Modellverwaltung

Der Server läuft rein lokal auf dem eigenen Rechner; nichts davon verlässt standardmäßig das Gerät.

Bionic zeigt in der geprüften Version unter **Settings > Local Models > Local Model API** Einstellungen für einen lokalen Modellzugriff (siehe Kapitel 7.1). Die vollständig dokumentierte REST-, OpenAI- und Anthropic-kompatible Serveroberfläche gehört zur klassischen LM-Studio-App beziehungsweise zu `llmster`. Für produktive Integrationen sollte deshalb die aktuelle Entwicklerdokumentation maßgeblich sein.

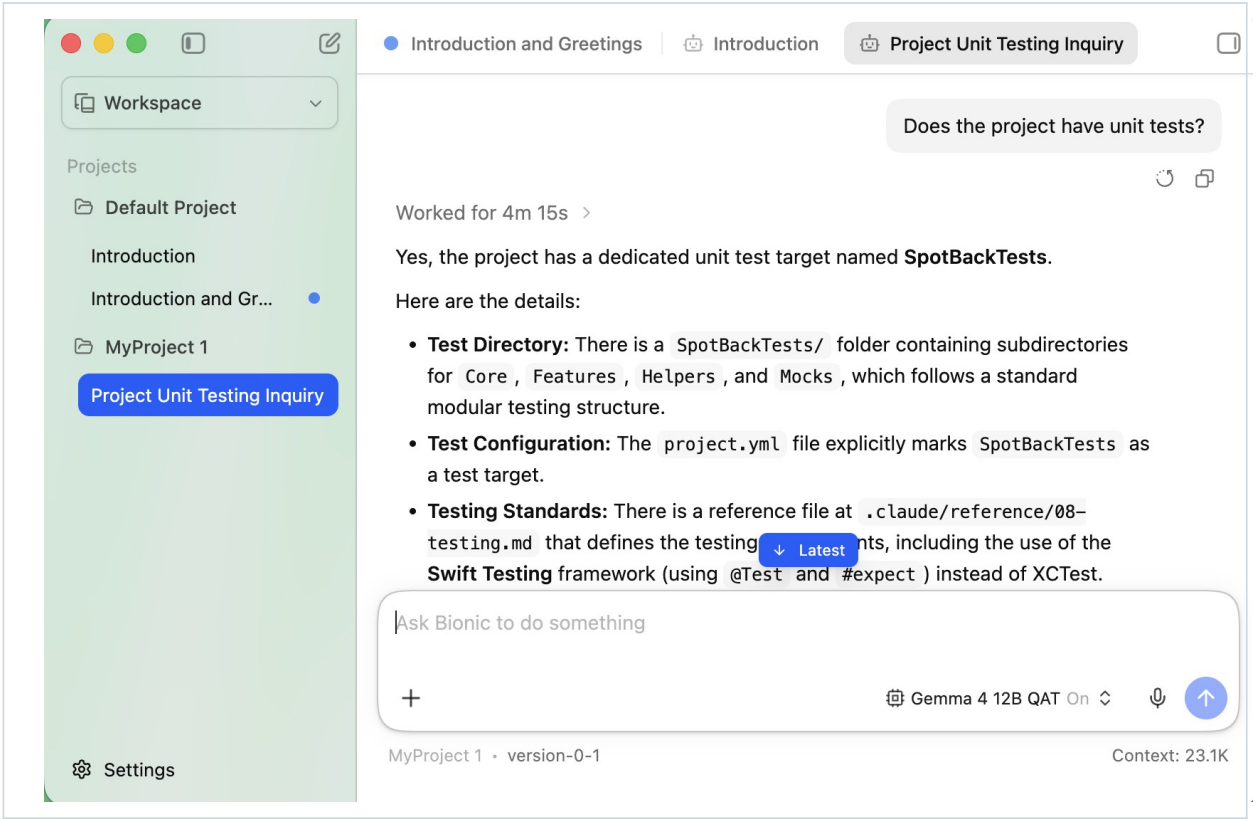
5.4 Erste Testfrage stellen (Funktionstest)

Nach dem ersten Modell-Download lohnt sich ein kurzer Funktionstest, bevor man in Bionic eine größere Aufgabe darauf aufbaut. In Bionic geschieht dies direkt in einer neuen Session:

1. Ein Projekt und darin eine neue Session öffnen
2. Im Model Picker prüfen, dass das gewünschte lokale Modell ausgewählt ist
3. Unten ins Eingabefeld eine einfache Testfrage tippen, z. B. „Wer bist du und was kannst du?" oder „Rechne 12 mal 7"

4. Mit Enter oder dem Senden-Button abschicken

Kommt innerhalb weniger Sekunden eine sinnvolle Antwort zurück, läuft das Modell lokal korrekt. Bionic kann jetzt im nächsten Kapitel darauf zugreifen. Bleibt die Antwort aus oder erscheint eine Fehlermeldung, liegt es meist entweder daran, dass noch kein Modell geladen ist (siehe oben, Schritt 4), oder der Arbeitsspeicher für die gewählte Quantisierungsstufe nicht ausreicht (siehe 5.1). Dann hilft eine kleinere Quantisierungsstufe oder ein kleineres Modell.



Erste Testfrage im Chat-Tab

6. Bionic einrichten: das erste Projekt

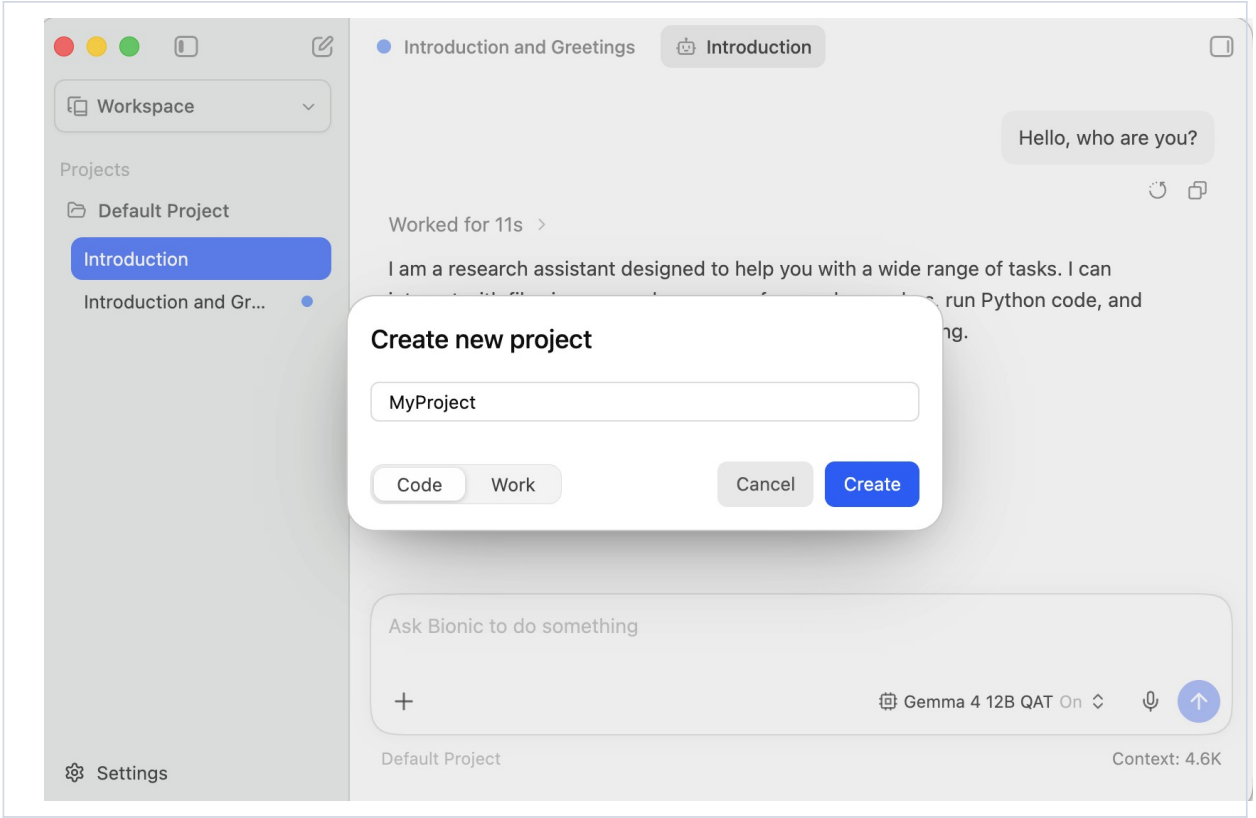
Bionic organisiert Arbeit in **Projekten**, vergleichbar mit Workspaces. Es gibt zwei Projekttypen:

- **Code Projects** – für die Arbeit an einem lokalen Code-Repository, inklusive Datei-, Such-, Git- und Shell-Werkzeugen
- **Work Projects** – für Recherche, Schreiben, Analyse und Dokumentenarbeit (PDFs, Präsentationen, Tabellen), ohne dass man zwingend einen Code-Ordner anhängen muss

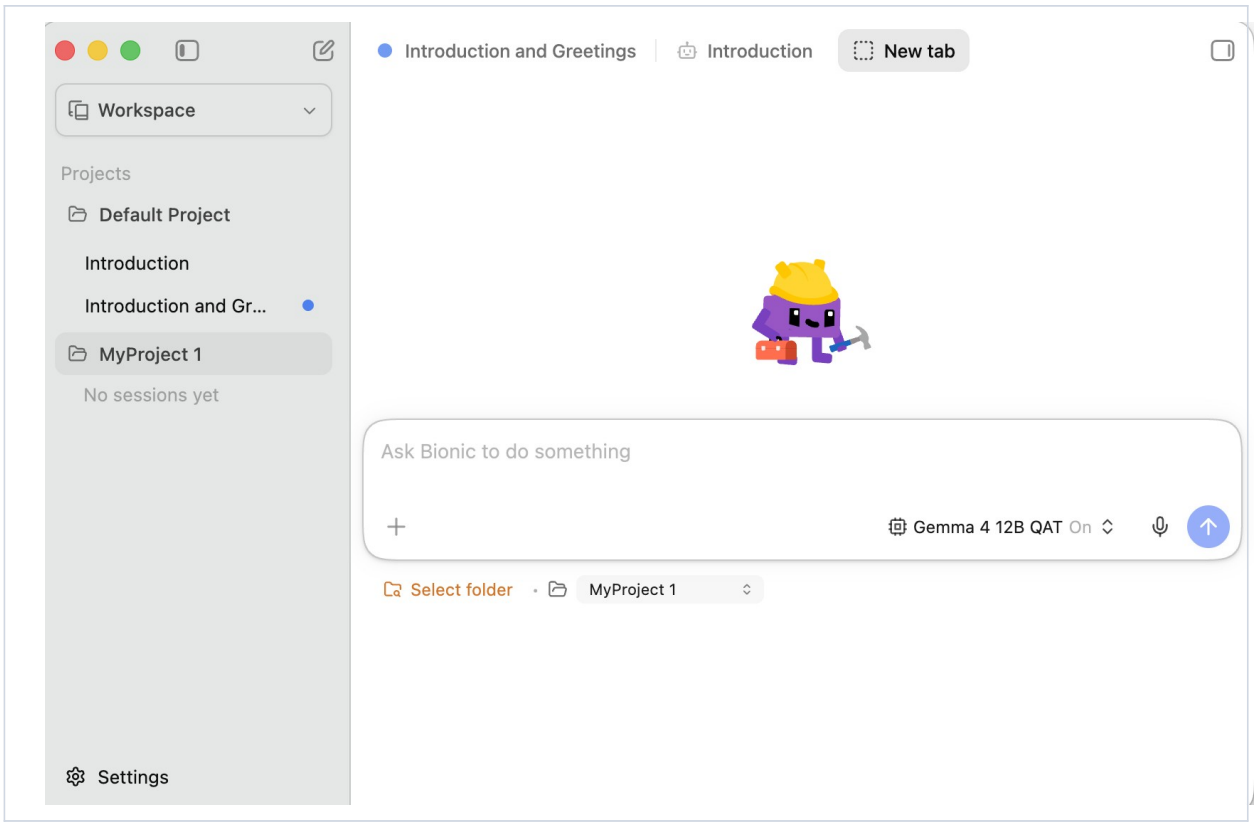
Ablauf für ein neues Code Project:

1. Bionic öffnen, „Neues Projekt" wählen, Typ „Code Project" auswählen
2. Projekt benennen
3. Modell auswählen – entweder ein in Bionic verfügbares lokales Modell, ein Remote-Modell über LM Link oder ein Cloud-Modell aus der LM Studio Secure Cloud
4. Einen lokalen Ordner anhängen (z. B. den Ordner eines bestehenden Projekts)
5. Bionic legt danach eine Session an, in der man in natürlicher Sprache Aufgaben stellt

Innerhalb eines Projekts lassen sich mehrere **Sessions** für unterschiedliche Teilaufgaben anlegen, ohne dass sich der Kontext gegenseitig stört.



Neues Projekt anlegen



Lokalen Ordner und Modell für das neue Projekt auswählen

7. Konfiguration

7.1 Die Settings-Oberfläche von Bionic im Detail (Mac, Version 1.0.1+1)

Anders als weiter oben zunächst beschrieben, hat Bionic auf dem Mac tatsächlich eine eigene, recht umfangreiche **Settings**-Oberfläche. Sie ist nur anders gegliedert als bei der klassischen LM-Studio-App. Die Bezeichnungen bleiben absichtlich englisch, weil die App selbst keine deutsche Oberfläche anbietet; erklärt wird hier auf Deutsch. Die Gliederung (Stand Version 1.0.1+1):

```
Settings
├── General
├── Sessions
├── Appearance
├── Voice
└── Billing and Usage
Integrations
└── Connected Apps
Devices
└── LM Link
Local Models
├── Explore
├── Library
├── Loaded Instances
├── Local Model API
├── Local Model Defaults
└── Runtime
```

Settings > General – Grundlegende App-Einstellungen und der Anmeldebereich: Hier meldet man sich mit dem (kostenlosen) LM-Studio-Konto an (Sign-in läuft über den Standardbrowser), das für Cloud-Modelle, LM Link und Connected Apps benötigt wird. Von hier aus verzweigt auch der Zugriff auf die Plan-Verwaltung („Manage Plan") weiter zu „Billing and Usage".

Settings > Sessions – Einstellungen rund um das Verhalten von Arbeits-Sessions. Die offizielle Dokumentation beschreibt die einzelnen Schalter dieses Bereichs bislang nicht vollständig. Deshalb sollten nur Optionen erklärt werden, die in der jeweils verwendeten Version tatsächlich sichtbar sind; Aussagen über Aufbewahrungsfristen oder Checkpoints wären ohne sichtbare Option spekulativ.

Settings > Appearance – Das Erscheinungsbild der App. Welche Optionen neben dem Farbschema angeboten werden, hängt von der installierten Preview-Version ab und sollte direkt anhand der Oberfläche beschrieben werden.

Settings > Voice – Ein- und Ausschalten sowie Konfiguration der lokalen Sprachdiktierfunktion (Voxtral-Modell, siehe Kapitel 7.5). Eine feste Downloadgröße sollte nur genannt werden, wenn Bionic sie in der aktuellen Version anzeigt.

Settings > Billing and Usage – Der komplette Abrechnungsbereich: „Manage Plan" leitet zum LM Studio Hub im Browser weiter, wo sich Cloud-Guthaben aufladen lässt. Unter „Billing account" wählt man, ob die Cloud-Nutzung dem eigenen, persönlichen Konto oder, falls vorhanden, einem Organisationskonto in Rechnung gestellt wird. Zusätzlich zeigt dieser Bereich den aktuellen Guthabenstand sowie Verbrauchsstatistiken der Cloud-Modelle an (siehe Kapitel 9, „Preise").

Integrations > Connected Apps – Hier werden in der geprüften Version externe Dienste verbunden. Ob dieser Bereich beliebige MCP-Server und dieselbe `mcp.json` wie die klassische App unterstützt, ist in der aktuellen Bionic-Dokumentation noch nicht vollständig beschrieben (siehe Kapitel 7.3).

Devices > LM Link – Verwaltung der Geräte-Kopplung: LM Link erlaubt es, ein lokal laufendes Modell von einem anderen eigenen Gerät aus zu nutzen, etwa ein leistungsstarkes Modell auf einem Desktop-Rechner laufen zu lassen, während man Bionic über ein MacBook bedient. In der kostenlosen Stufe nennt die Preisseite bis zu fünf Geräte (siehe Kapitel 9, „Preise").

Local Models > Explore – Der Modell-Download-Bereich, das Gegenstück zum Such-/Download-Bereich der klassischen LM-Studio-App: Modelle nach Namen suchen, nach Format filtern und, weil ein „Device-Fit"-Hinweis direkt mit angezeigt wird, sofort sehen, ob ein Modell zur eigenen Hardware passt. Sind über LM Link mehrere Geräte gekoppelt, lässt sich hier zusätzlich auswählen, auf welches Gerät heruntergeladen werden soll.

Local Models > Library – Die Übersicht aller bereits heruntergeladenen, auf dem Gerät indizierten Modelle (das Gegenstück zu „My Models" in der klassischen App). Von hier aus wählt man beim Start einer Session, welches lokale Modell verwendet werden soll. Das Löschen nicht mehr benötigter Modelle (siehe Kapitel 5.2) läuft ebenfalls über diese Ansicht.

Local Models > Loaded Instances – Zeigt, welche Modelle gerade tatsächlich im Arbeitsspeicher/der GPU geladen sind (nicht nur heruntergeladen, sondern aktiv bereit für Anfragen), inklusive ihrer aktuellen Lade-Konfiguration (u. a. Kontextlänge, Batch-Größe, wie viele parallele Anfragen möglich sind). Das ist das Bionic-Gegenstück zur Runtime-Verwaltung der klassischen App (dort per Tastenkürzel `⌘ Shift R` erreichbar). Hier lässt sich ein nicht mehr benötigtes Modell auch gezielt wieder aus dem Speicher entladen, ohne es von der Festplatte zu löschen.

Local Models > Local Model API – Der bereits beschriebene lokale, OpenAI-kompatible Server für eigene Skripte/Programme (siehe Kapitel 5.3): Ein- und Ausschalten des Servers, Port-Nummer, ob auch andere Geräte im selben Netzwerk zugreifen dürfen, und ob ein Zugriffstoken verlangt wird. Inhaltlich derselbe Baustein wie die „Server"-Einstellungen der klassischen LM-Studio-App, hier nur unter „Local Models" statt in einem eigenen Server-Tab einsortiert.

Local Models > Local Model Defaults – Voreinstellungen für lokale Modelle. Welche Parameter dort global oder pro Modell gelten, sollte anhand der aktuellen Oberfläche geprüft werden; die ausführliche Dokumentation zu „Per-model Defaults" bezieht sich auf die klassische LM-Studio-App und lässt sich nicht automatisch auf Bionic übertragen.

Local Models > Runtime – Verwaltung der eigentlichen Ausführungs-Engines, die im Hintergrund für lokale Modelle sorgen: `llama.cpp` (für GGUF-Modelle) und Apples `MLX` (auf Apple-Silicon-Macs für MLX-Modelle). Hier lässt sich in der Regel ablesen, welche Engine-Version aktuell installiert ist, und ein Update der Runtime anstoßen, ohne gleich die ganze App neu installieren zu müssen.

> **Wichtig zur Einordnung:** Für mehrere Einstellungsbereiche liegt noch keine ausführliche, punktgenaue Bionic-Dokumentation vor. Die Beschreibungen oben trennen deshalb zwischen direkt beobachtbarer Oberfläche und offiziell dokumentiertem Verhalten. Konzepte der klassischen LM-Studio-App dürfen nicht ohne Weiteres auf Bionic übertragen werden. Gut dokumentiert sind derzeit vor allem Projekte und Sessions, Modellwahl, lokale Downloads sowie Konten und Abrechnung (siehe Quellen).

7.2 Presets (System Prompts & Parameter)

Presets stammen aus der **klassischen LM-Studio-App** (nicht aus Bionic selbst) und bündeln System Prompt sowie Inferenz-Parameter (zum Beispiel Temperature und Top P) in einer wiederverwendbaren, benannten Konfiguration, vergleichbar mit einem gespeicherten System-Prompt-Profil. Load-Parameter sind laut aktueller Preset-Dokumentation nicht Bestandteil des neuen Preset-Formats; dafür empfiehlt LM Studio „Per-model Defaults".

Muss ich die klassische LM-Studio-App dafür extra installieren? Für das Herunterladen und Verwenden lokaler Modelle reicht Bionic aus (siehe Kapitel 5). Wer klassische LM-Studio-Presets anlegen und

verwalten möchte, benötigt dafür derzeit die klassische App. Wichtig ist jedoch: Die aktuelle Bionic-Dokumentation beschreibt weder eine eigene Preset-Verwaltung noch die Auswahl dieser klassischen Presets in Bionic. Ein in LM Studio erstelltes Preset sollte daher nicht als automatisch wirksame Bionic-Konfiguration dargestellt werden.

Wo liegen Presets auf der Festplatte?

Plattform	Pfad
macOS/Linux	`~/.lmstudio/config-presets/`
Windows	`%USERPROFILE%\.lmstudio\config-presets`

Jedes Preset liegt als eigene, lesbare `.json`-Datei in diesem Ordner. *(Korrektur gegenüber älteren Angaben im Netz: Vereinzelt kursiert ein abweichender Pfad wie `~/.cache/lm-studio/model-presets.json` – laut aktueller offizieller Dokumentation ist der oben genannte Ordner-Pfad der korrekte, aktuelle Speicherort.)*

Konkretes Beispiel – ein eigenes Preset „Swift-Assistent" anlegen:

1. Die **klassische LM-Studio-App** öffnen (nicht Bionic) und ein beliebiges Modell laden
2. Im Chat-Fenster rechts die Konfigurations-Seitenleiste öffnen (Regler-/Zahnrad-Symbol)
3. Oben in dieser Seitenleiste zeigt ein Dropdown das aktuell aktive Preset an, standardmäßig „Default" bzw. „unsaved preset"
4. In das Feld „System Prompt" einen eigenen Text eintragen, z. B.: „Du bist ein erfahrener Swift-Entwickler. Antworte auf Deutsch, kurz und ohne Floskeln, Code-Beispiele bleiben auf Englisch."
5. Bei Bedarf Parameter in der „Advanced Configuration" anpassen, z. B. die Temperature für konsistentere Code-Antworten von 0.8 auf 0.3 senken
6. Auf **„Save Preset"** klicken – ein Dialog fragt nach einem Namen, z. B. „Swift-Assistent"; bestätigen
7. Das neue Preset erscheint danach im Dropdown und lässt sich für jeden künftigen Chat auswählen, ohne den System Prompt jedes Mal neu eintippen zu müssen

Bearbeiten: Preset im Dropdown auswählen, Werte ändern, erneut „Save Preset" wählen. Bei eigenen Presets überschreibt das den bestehenden Stand. Für die volle Kontrolle lässt sich jede `.json`-Datei im oben genannten Ordner auch direkt in einem Texteditor bearbeiten, da die Dateistruktur lesbar ist.

Löschen: über den Preset-Manager in der Seitenleiste (Drei-Punkte-/Papierkorb-Menü neben dem jeweiligen Eintrag) oder, analog zum manuellen Modell-Löschen in Kapitel 5.2, durch schlichtes Entfernen der zugehörigen `.json`-Datei aus `config-presets`.

Muss ein Preset in Bionic irgendwo „angesprochen" oder registriert werden? Dafür ist derzeit kein offizieller Ablauf dokumentiert. Presets werden in der klassischen LM-Studio-App verwaltet und dort Chats zugeordnet. Eine Bionic-Auswahl oder eine feste Zuordnung zu einem Bionic-Projekt ist in der aktuellen Dokumentation nicht beschrieben. Kapitel 14 verwendet Presets deshalb nicht mehr als Ersatz für ein projektweites Regelwerk.

Presets lassen sich zudem als Datei teilen oder im „LM Studio Hub" veröffentlichen. Das ist nützlich, um ein einmal gebautes Preset zwischen mehreren eigenen Geräten oder mit anderen zu teilen.

7.3 MCP-Server und Connected Apps

In der geprüften Bionic-Version ist unter **Integrations** ein Bereich **Connected Apps** sichtbar. Die aktuelle offizielle Bionic-Dokumentation erklärt dessen vollständigen Funktionsumfang jedoch noch nicht. Deshalb dürfen die ausführlich dokumentierten MCP-Schritte der klassischen LM-Studio-App nicht automatisch als Bionic-Anleitung ausgegeben werden.

Für die **klassische LM-Studio-App** ist folgender MCP-Ablauf offiziell dokumentiert:

1. In der rechten Seitenleiste den Tab „Program” öffnen
2. Unter „Install” auf „Edit mcp.json” klicken
3. Den gewünschten MCP-Server als JSON-Eintrag ergänzen
4. Alternativ, wo verfügbar, den „Add to LM Studio”-Knopf eines MCP-Anbieters verwenden

Ob und wie derselbe Server anschließend in Bionic verfügbar ist, sollte in der installierten Bionic-Version geprüft werden. MCP-Server können auf lokale Dateien, Netzwerkdienste oder vertrauliche Daten zugreifen. Deshalb sollten ausschließlich vertrauenswürdige Server installiert und deren Berechtigungen sorgfältig geprüft werden.

7.4 Sandbox und Checkpoints

Bionic arbeitet mit einer Sandbox-Umgebung für die Dateiverwaltung sowie automatischen Checkpoints, über die sich vorgenommene Änderungen rückgängig machen lassen. Praktisch ist das ein eingebautes Undo auf Ebene ganzer Arbeitsschritte, nicht nur einzelner Tastatureingaben.

7.5 Sprachsteuerung (Voice)

Bionic bringt eine systemweite Diktierfunktion mit, die zum Launch auf Mistral AIs **Voxtral**-Modell basiert und komplett lokal auf dem Gerät läuft (keine Cloud-Transkription). Sie funktioniert app-übergreifend: Man kann den Cursor in ein Textfeld setzen und dort diktieren, nicht nur innerhalb von Bionic. Der zusätzliche Speicherbedarf sollte direkt in der aktuellen App-Version geprüft werden; die offizielle Ankündigung nennt keine feste Downloadgröße.

8. Lokal oder Cloud – die zwei Ausführungsmodi

Bionic unterscheidet konsequent zwei Wege, ein Modell auszuführen:

- **Lokal:** über die LM-Studio-Runtime auf dem eigenen Rechner; nach dem Download offline-fähig, solange keine Websuche, Connected App oder andere Netzwerkfunktion verwendet wird
- **LM Studio Secure Cloud:** für rechenintensivere Frontier-Modelle, die lokal (noch) nicht praktikabel liefen – mit „Zero Data Retention" und laut Hersteller transienter (nicht dauerhafter) Datenverarbeitung
- **LM Link:** erlaubt es, ein lokal laufendes Modell auch von anderen eigenen Geräten aus anzusprechen (in der kostenlosen Stufe für bis zu 5 Geräte) – kein Cloud-Modell, sondern Fernzugriff auf die eigene lokale Instanz

Für welchen Weg man sich in einem Projekt entscheidet, lässt sich pro Session bei der Modellauswahl festlegen. Man ist also nicht auf eine feste Wahl pro Projekt festgelegt.

9. Preise

Stufe	Preis	Enthalten
Free	0 $	Bionic-Agent, lokale LLMs über llama.cpp/MLX, Offline-Sprachtranskription, ZDR-Websuche (begrenztes Kontingent), LM Link für bis zu 5 Geräte, keine Datenspeicherung auf fremden Servern
Pay-as-you-go (Cloud Credits)	nach Verbrauch, konkrete Preise nicht veröffentlicht	Zugriff auf Frontier-Cloud-Modelle wie GLM 5.2, Kimi K2.6, Kimi Code K2.7, DeepSeek V4 Pro, Rechenzentrum in den USA, ZDR als Standard-Datenschutzrichtlinie
Bionic Pass	„Pricing and plan details coming soon" (Stand Juli 2026 noch nicht veröffentlicht)	Leistungsumfang und Ausgestaltung sind noch nicht bekannt

Wichtig: Die App und die lokale Nutzung eigener Modelle sind nach dem derzeit veröffentlichten Preismodell kostenlos. Kosten entstehen bei der Nutzung kostenpflichtiger Cloud-Inferenz. Aussagen wie „bleiben kostenlos" wären eine nicht belegbare Zusage für die Zukunft.

10. Möglichkeiten: Was kann man mit Bionic tun?

Code Projects: - Bestehenden Code lesen, erklären und gezielt ändern - „Agentic Code Search" – eigenständiges, iteratives Durchsuchen eines Repositories nach relevanten Stellen, statt nur den aktuell geöffneten Dateiausschnitt zu kennen - Inline-Diffs zur Überprüfung von Änderungen, bevor sie endgültig übernommen werden - Zugriff auf Git (Commits, Historie) sowie eine Shell für Build-/Testbefehle

Work Projects: - Arbeiten mit PDFs, Präsentationen, Tabellenkalkulationen - Verzeichnisse organisieren, Inhalte zusammenfassen - Websuche einbinden (im Free-Tier mit begrenztem Kontingent) - Automatische Checkpoints zum Zurückspringen auf einen früheren Stand

Plattformübergreifend: - Lokale Sprachdiktierfunktion in beliebigen Apps - Wahlfreiheit zwischen komplett lokaler Verarbeitung und Cloud-Modellen, je nach Aufgabe und Datenschutzbedarf

11. Arbeiten mit Xcode (Beispiel)

Bionic hat, Stand Juli 2026, **kein natives Xcode-Plugin** und keine offizielle IDE-Integration. Die Anbindung läuft, ähnlich wie bei anderen dateibasierten Coding-Agenten, indirekt über das Dateisystem:

1. In Bionic ein neues **Code Project** anlegen und den Ordner des Xcode-Projekts als lokalen Ordner anhängen
2. Ein passendes Modell wählen – für ein reines Swift-Kommandozeilenprojekt reicht zum Ausprobieren ein kleines lokales Modell (z. B. Gemma 4), für „echte" SwiftUI-/UIKit-Multi-File-Arbeit empfiehlt sich ein stärkeres Modell (Qwen 3.6 35B-A3B lokal oder ein Cloud-Modell wie GLM 5.2/Kimi K2.7 Code)
3. Xcode parallel geöffnet lassen – Bionic ändert Dateien direkt auf der Festplatte, Xcode erkennt externe Änderungen automatisch und lädt sie neu
4. Aufgabe in natürlicher Sprache formulieren, z. B.: „Füge der Struct `Calculator` eine Methode `subtract` hinzu, die zwei `Int`-Werte subtrahiert, und schreibe dazu einen passenden `XCTest`."
5. Bionic durchsucht das angehängte Repository (Agentic Code Search), schlägt Änderungen als Inline-Diff vor und kann beispielsweise `swift test` über die Shell ausführen
6. Diffs und Befehlsausgaben prüfen, Änderungen übernehmen und das Ergebnis in Xcode beziehungsweise im Test-Navigator kontrollieren
7. Bei Bedarf über den automatischen Checkpoint auf den Stand vor der Änderung zurückspringen

> **Praxis-Hinweis aus unabhängigen Tests:** Kleine, lokale Modelle (2B–4B) liefern bei überschaubaren, klar umrissenen Aufgaben brauchbare Ergebnisse, tun sich aber bei größeren SwiftUI-Projekten mit vielen zusammenhängenden Dateien noch schwer, etwa bei der korrekten Verdrahtung mehrerer Komponenten untereinander. Für ernsthafte Xcode-Projekte lohnt sich entweder ein deutlich größeres lokales Modell (wenn die Hardware es hergibt) oder der Wechsel auf ein Cloud-Modell der LM Studio Secure Cloud für die jeweilige Session.

12. Vergleich mit Claude Code

Kriterium	LM Studio Bionic	Claude Code
Modellbindung	Offene Modelle lokal sowie ausgewählte Cloud-Modelle (GLM, Kimi, DeepSeek)	Nativ Claude-Modelle; über LM Studios Anthropic-kompatible API sind auch lokale Modelle möglich
Bedienung	Eigenständige Desktop-App (GUI), projektbasiert	Terminal/CLI-Agent, dateibasiert im Repo
Reifegrad	Sehr neu (Launch Mitte Juli 2026), früher Produktstand	Deutlich länger am Markt, ausgereifter
Lokale Ausführung	Zentrales Feature, Kernversprechen der App	Über einen kompatiblen lokalen Endpunkt möglich; LM Studio dokumentiert dies seit Version 0.4.1
Preismodell	App kostenlos, Cloud-Nutzung nach Verbrauch (Pay-as-you-go), Abo „Bionic Pass" in Vorbereitung	Abo-/API-Modell direkt bei Anthropic
Sprachsteuerung	Ja, lokal (Voxtral), app-übergreifend	Keine direkt vergleichbare integrierte systemweite Voice-Tastatur dokumentiert
Tool-Sicherheit	Inline-Diffs und Checkpoints; konkrete Freigaben abhängig von App-Version und Werkzeug	Granulares Permission-System (allow/deny/ask-Regeln), zusätzlich Hooks
Erweiterbarkeit	Connected Apps sichtbar; vollständiger Bionic-MCP-Ablauf noch nicht ausführlich dokumentiert	MCP-Server, plus eigenes Skill-/Subagenten-/Hook-System
Fokus	Datenschutzbewusste, lokale Agentenarbeit mit offenen Modellen	Tiefe, hochintegrierte Agenten-Workflows im Anthropic-Ökosystem

Bionic punktet dort, wo Datenhoheit, lokale Ausführung und eine grafische Projektoberfläche im Vordergrund stehen. Claude Code punktet bei komplexen, mehrstufigen Coding-Aufgaben, granularer Steuerung des Agentenverhaltens und einem ausgereiften Ökosystem an Erweiterungsmechanismen (siehe Kapitel 13). Mit LM Studios Anthropic-kompatibler API lässt sich Claude Code ebenfalls gegen ein lokales Modell betreiben; Qualität und Kompatibilität hängen dann vom gewählten Modell ab.

13. Rules, Hooks, Commands, Skills – gibt es das bei Bionic?

Das ist der Bereich, in dem Bionic aktuell am deutlichsten hinter Claude Code zurückliegt. Eine ehrliche Gegenüberstellung, Stand Juli 2026:

Claude-Code-Konzept	Zweck bei Claude Code	Entsprechung bei Bionic
`CLAUDE.md` / `.claude/rules/*.md`	Projektspezifische, dauerhaft geltende Regeln und Konventionen	**Kein direktes, offiziell dokumentiertes Äquivalent gefunden.** Klassische LM-Studio-Presets sind keine belegte Bionic-Lösung
`.claude/settings.json` (Hooks)	Automatisierte Aktionen bei bestimmten Ereignissen (z. B. „nach jedem Edit automatisch formatieren")	**Kein Hook-System dokumentiert.** Inline-Diffs und Checkpoints helfen bei der Kontrolle, ersetzen aber keine Ereignis-Hooks
`.claude/commands/*.md` (Slash Commands)	Selbst definierte, wiederverwendbare Befehlsabkürzungen	**Nicht dokumentiert.** Bionic wird über natürliche Sprache in Sessions bedient, kein Hinweis auf ein Custom-Command-System
`.claude/skills/*` (Skills)	Gekapselte, bei Bedarf ladbare Fähigkeiten/Anleitungen für wiederkehrende Aufgabentypen	**Kein Äquivalent.** Am nächsten kommt die feste Unterscheidung zwischen Code Project und Work Project als zwei grob vordefinierte „Modi"
MCP-Server	Anbindung externer Werkzeuge/Datenquellen über das Model Context Protocol	**Für die klassische LM-Studio-App dokumentiert.** Bionics „Connected Apps" sind noch nicht gleichwertig ausführlich dokumentiert (siehe Kapitel 7.3)
Permission-Regeln (allow/deny/ask)	Feingranulare, dauerhaft gespeicherte Freigaberegeln pro Werkzeug/Befehl	**Kein gleichwertiges System in der aktuellen Bionic-Dokumentation beschrieben.** Die konkrete Oberfläche der installierten Preview-Version prüfen

Fazit: Bei dem, was Claude Code über `.claude`/`CLAUDE.md` an dauerhafter, projektspezifischer Anpassung bietet (Regeln, Hooks, eigene Befehle und Skills), hat Bionic zum jetzigen Zeitpunkt keine gleichwertig dokumentierte Entsprechung. Wer feste Projektregeln nutzen will, sollte sie als versionierte Datei im Repository pflegen und Bionic zu Beginn einer Session ausdrücklich zum Lesen dieser Datei auffordern. Ein klassisches LM-Studio-Preset ist dafür kein belegter automatischer Mechanismus. Da Bionic erst wenige Tage alt ist, kann sich dieser Bereich schnell verändern.

14. Umstieg von Claude Code: wie arbeitet man jetzt?

Wer bisher mit Claude Code gearbeitet hat, muss beim Wechsel zu bzw. bei der Ergänzung um Bionic vor allem eine mentale Umstellung vornehmen: von einem terminalzentrierten Agenten mit automatisch geladenem Regelwerk und harness-erzwungenen Hooks hin zu einem GUI-Agenten, bei dem so gut wie jede projektspezifische Anpassung **manuell nachgebaut und manuell aufgerufen** werden muss. Nichts davon läuft „von selbst" im Hintergrund, wie man es von `.claude/settings.json` gewohnt ist.

Die praktikabelste Strategie: die eigenen Claude-Code-Artefakte (CLAUDE.md, `.claude/rules/*.md`, Skills, Commands) nicht wegwerfen, sondern als Rohmaterial für die unten beschriebenen Bionic-Ersatzkonstruktionen weiterverwenden. Der Inhalt bleibt meist eins zu eins gültig, nur der Auslöse-Mechanismus fehlt und muss ersetzt werden.

14.1 Rules nachbauen: eine Projektdatei ausdrücklich einlesen lassen

`AGENTS.md` ist ein verbreitetes Format für projektspezifische Agentenanweisungen. Ob Bionic diese Datei automatisch erkennt und bei jeder Anfrage lädt, ist in der aktuellen Dokumentation nicht bestätigt. Sie ist trotzdem eine praktische, versionierbare Ablage für Projektregeln.

Ein belastbarer manueller Ablauf sieht so aus:

1. `AGENTS.md` im Projektstamm anlegen; der Inhalt kann weitgehend aus einer bestehenden `CLAUDE.md` übernommen werden.
2. Die Regeln kurz, eindeutig und überprüfbar formulieren: Build-Befehle, Testanforderungen, Stilregeln und verbotene Änderungen.
3. Zu Beginn jeder neuen Bionic-Session ausdrücklich schreiben: „Lies zuerst `AGENTS.md` und beachte diese Regeln während der gesamten Session."
4. Bei wichtigen Aufgaben die entscheidenden Einschränkungen zusätzlich in der konkreten Aufgabe wiederholen.

Das ist weniger automatisch als `CLAUDE.md` in Claude Code. Ein Preset der klassischen LM-Studio-App ist kein dokumentierter Auto-Loader für Bionic und wird deshalb nicht als Lösung empfohlen.

14.2 Skills nachbauen: ein `playbooks/`-Ordner statt `.claude/skills/`

Ein automatisches Nachladen einzelner Skills je nach Aufgabe gibt es nicht. Der pragmatische Ersatz:

1. Im Projekt einen Ordner `playbooks/` (oder `skills/`) anlegen, darin eine Markdown-Datei je wiederkehrendem Workflow – z. B. `playbooks/code-review.md`, `playbooks/release-checklist.md`, `playbooks/neue-view-anlegen.md` – Inhalt entspricht dem, was vorher als Claude-Code-Skill gepflegt wurde
2. In der Session gezielt darauf verweisen: „Folge der Anleitung in `playbooks/code-review.md` für diese Änderung."
3. Bei häufig gebrauchten Workflows eine kurze Startformulierung in `prompts/` ablegen, die ausdrücklich auf das jeweilige Playbook verweist

Der Unterschied zu Claude Code: Es gibt keine automatische Erkennung „diese Aufgabe passt zu Skill X". Die Auswahl trifft man selbst, jedes Mal aufs Neue.

14.3 Commands nachbauen: Textbausteine statt Slash Commands

Ein eigenes Befehlssystem mit Kurznamen und Parametern (`$ARGUMENTS`) existiert nicht. Zwei Ersatzwege:

- Ein Ordner `prompts/` mit fertig formulierten Aufgabenvorlagen als einfache Textdateien, die man bei Bedarf in die Session hineinkopiert
- Wiederkehrende, immer gleiche Aufgabentypen (z. B. „Commit-Message nach Konvention X formulieren") als eigene Prompt-Datei ablegen und in der Session einfügen oder referenzieren

Beides ist reine Copy-Paste-Disziplin, keine Automatisierung.

14.4 Hooks nachbauen: die Automatisierung wandert aus der App heraus

Das ist die größte strukturelle Lücke. Da es keinen PostToolUse-/PreToolUse-Mechanismus gibt, muss alles, was bei Claude Code ein Hook erledigt hätte, entweder außerhalb von Bionic laufen oder als Bitte an das Modell selbst formuliert werden:

- **Projekteigene Automatisierung nutzen statt App-Hooks:** Git-Pre-Commit-/Pre-Push-Hooks, Xcode-Build-Phasen, ein Makefile-Target oder eine CI-Pipeline übernehmen das, was in Claude Code z. B. „nach jedem Edit automatisch `swiftlint` laufen lassen" war. Diese Mechanismen greifen unabhängig davon, welches KI-Tool gerade am Code gearbeitet hat. Dadurch sind sie sogar robuster als App-gebundene Hooks
- **Den Agenten per Regel zur Selbstkontrolle anhalten:** In `AGENTS.md` explizit festhalten, dass nach jeder Code-Änderung `swift test` beziehungsweise das passende Build-/Testkommando auszuführen ist, bevor die Aufgabe als erledigt gilt. Das ist keine erzwungene Garantie wie ein echter Hook, sondern eine Anweisung an das Modell.
- **Änderungen bewusst prüfen:** Inline-Diffs, Shell-Ausgaben und Git-Diff kontrollieren. Welche Freigabedialoge Bionic anzeigt, hängt von Version und Werkzeug ab; die aktuelle Dokumentation verspricht keinen identischen Dialog vor ausnahmslos jedem Aufruf.

14.5 Permission-Regeln: kein gleichwertiges System dokumentiert

Die aktuelle Bionic-Dokumentation beschreibt kein mit Claude Code vergleichbares, dauerhaftes `allow`/`deny`/`ask`-Regelwerk. Daraus folgt nicht automatisch, dass jede App-Version jeden Werkzeugaufruf auf dieselbe Weise bestätigt. Die tatsächlich angebotenen Freigabeoptionen sollten in der installierten Preview-Version geprüft und sicherheitskritische Änderungen zusätzlich über Git kontrolliert werden.

14.6 MCP-Server: Konfiguration nicht ungeprüft übernehmen

MCP ist ein offener Standard, die konkrete Konfiguration und Authentifizierung kann sich zwischen Clients aber unterscheiden. Bereits für Claude Code eingerichtete Server sind daher ein guter Ausgangspunkt, sollten jedoch nicht ungeprüft als Bionic-`mcp.json` übernommen werden. Kapitel 7.3 trennt die offiziell dokumentierte LM-Studio-Konfiguration vom bislang nur teilweise dokumentierten Bionic-Bereich „Connected Apps".

14.7 Konkreter Projektaufbau als Vorlage

Für ein Projekt, das vorher nach dem Claude-Code-Muster organisiert war, hat sich folgende Struktur als Bionic-Äquivalent bewährt:

```
mein-projekt/
├── AGENTS.md                  # zentrale, versionierte Projektregeln
│                               #   → zu Beginn der Bionic-Session
ausdrücklich lesen lassen
├── playbooks/                 # ersetzt .claude/skills/
│   ├── code-review.md
│   └── release-checklist.md
├── prompts/                   # ersetzt .claude/commands/
│   └── commit-message.md
└── .git/hooks/                # ersetzt .claude/settings.json-Hooks
    └── pre-commit
```

Zusammengefasst als Übersicht:

Claude-Code-Baustein	Bionic-Nachbau	Automatisierungsgrad
`CLAUDE.md` / `.claude/rules/*.md`	`AGENTS.md` oder vergleichbare Projektdatei, zu Sessionbeginn ausdrücklich lesen lassen	Niedrig bis mittel – manueller Startschritt
`.claude/skills/*`	`playbooks/*.md`, manuell referenziert	Niedrig – manuelle Auswahl nötig
`.claude/commands/*.md`	`prompts/*.md` zum Einfügen oder Referenzieren	Niedrig – keine dokumentierten Custom Commands
Hooks (`settings.json`)	Git-Hooks/CI/Build-Phasen außerhalb von Bionic + Regel, Tests auszuführen	Mittel – App-unabhängig, aber nicht App-erzwungen
Permission-Regeln (allow/deny)	Kein gleichwertiges dokumentiertes Regelsystem	Abhängig von der aktuellen App-Version
MCP-Server	Serverdaten als Ausgangspunkt verwenden, Client-Konfiguration prüfen	Abhängig von Integration und Authentifizierung

Damit lässt sich keine exakte Kopie der Claude-Code-Arbeitsumgebung erzeugen. Feste Projektregeln, wiederverwendbare Workflows und automatisierte Qualitätssicherung bleiben möglich, sind aber auf Bionic, versionierte Projektdateien und externe Automatisierung verteilt. Insbesondere das automatische Laden der Regeln ist ohne dokumentierte Bionic-Unterstützung nicht garantiert.

15. Datenschutz und Sicherheit

- **Zero Data Retention:** Für alle Bionic-Nutzer verspricht der Hersteller, keine Daten zu speichern oder für Training zu verwenden. Laut eigener Aussage gilt das ausdrücklich auch für die Cloud-Funktionen
- **Lokale Verarbeitung als Standard:** Reine lokale Inferenz und Sprachtranskription laufen auf dem Gerät. Aktivierte Websuche, Cloud-Modelle oder Connected Apps können dagegen Daten an externe Dienste übertragen.
- **Sandbox:** Dateizugriffe laufen über eine Sandbox-Umgebung
- **Änderungskontrolle:** Bionic zeigt Arbeitsergebnisse, Inline-Diffs und Checkpoints zur Prüfung beziehungsweise zum Zurückrollen. Die aktuelle Bionic-Dokumentation bestätigt jedoch nicht pauschal einen identischen Bestätigungsdialog vor jedem einzelnen Datei-, Shell- und MCP-Aufruf.

16. Vorteile und Nachteile

Vorteile:

- Starker Datenschutz-Fokus, lokale Ausführung als echtes Kernfeature, nicht nur Marketing-Zusatz
- Kostenlose Basisnutzung, keine Pflicht zu einem Cloud-Konto für rein lokale Arbeit
- Eingebaute, komplett lokale Sprachsteuerung app-übergreifend
- Bereich für Connected Apps; die klassische LM-Studio-App unterstützt zusätzlich MCP nach offenem Standard
- Freie Wahl zwischen lokalen und Cloud-Modellen, je nach Aufgabe und Datenschutzbedarf

Nachteile:

- Sehr frühes Produktstadium (wenige Tage am Markt zum Zeitpunkt dieser Anleitung), entsprechend rau an einigen Ecken
- Kein natives Xcode-Plugin dokumentiert; Integration erfolgt über den lokalen Projektordner
- Kleine, lokal praktikable Modelle sind bei komplexen Multi-File-Refactorings in größeren Codebasen noch schwach
- Kein Regel-/Hook-/Skill-System vergleichbar mit Claude Code – Projektanpassung ist aktuell deutlich eingeschränkter
- „Bionic Pass”-Preise noch nicht veröffentlicht, dadurch schwer langfristig kalkulierbar
- RAM-Druck möglich, wenn Agent, lokales Modell und Sprachfunktion gleichzeitig aktiv sind

17. FAQ

17.1 Lassen sich auch bereits vorhandene Ollama-Modelle nutzen?

Nicht automatisch, aber je nach Ausgangsformat mit zusätzlichem Aufwand. Sowohl Ollama als auch LM Studio können GGUF-Modelle über llama.cpp ausführen; 2026 unterstützen beide auf Apple Silicon außerdem MLX in unterschiedlichem Umfang. Ihre Modellverwaltung und Ablagestruktur unterscheiden sich:

	Ollama	LM Studio
Mac (Standardpfad)	`~/.ollama/models`	`~/.lmstudio/models/` (ältere Installationen teils `~/.cache/lm-studio/models`)
Windows (Standardpfad)	`C:\Users\<Name>\.ollama\models`	`C:\Users\<Name>\.lmstudio\models` (ältere Installationen teils `C:\Users\<Name>\.cache\lm-studio\models`)
Format auf der Platte	inhaltsadressierte Blobs (SHA-256-Namen), referenziert über eigene Manifest-Dateien – keine erkennbaren `.gguf`-Dateinamen	Hugging-Face-artige Ordnerstruktur `publisher/modell/modell-datei.gguf`, Dateinamen bleiben lesbar
Pfad änderbar?	ja, über die Umgebungsvariable `OLLAMA_MODELS` bzw. einen Symlink/Junction auf den Standardordner	ja, direkt in LM Studio unter Einstellungen bzw. im „My Models"-Tab

Der Ordner beginnt bei beiden Tools mit einem Punkt und ist damit ein verstecktes Verzeichnis. Unter Windows müssen dafür in den Explorer-Optionen „Ausgeblendete Elemente" eingeblendet werden, unter macOS hilft im Finder die Tastenkombination `⌘ Umschalt .`.

Weil Ollama seine Modelle als Blobs ohne `.gguf`-Endung ablegt, durchsucht LM Studios eigener Modell-Browser diesen Ordner nicht automatisch. Ein bereits über Ollama heruntergeladenes Modell taucht also nicht von selbst in LM Studio oder Bionic auf. Zwei Wege, um trotzdem dieselbe, bereits heruntergeladene Datei weiterzuverwenden, statt sie doppelt herunterzuladen:

1. **Offizieller, experimenteller Weg über die LM-Studio-CLI:** `lms import <pfad/zur/datei.gguf>` und dem interaktiven Prompt folgen. LM Studio erwartet danach eine Verzeichnisstruktur wie bei Hugging-Face-Downloads: `~/.lmstudio/models/publisher/modell/datei.gguf`
2. **Community-Tools zum automatischen Verlinken:** Skripte wie `llamalink` oder `ollama-to-lmstudio-symlinks` durchsuchen die Ollama-Manifeste selbstständig und legen passend benannte Symlinks im LM-Studio-Modellordner an. Beide Programme greifen danach auf dieselbe Datei zu, ohne dass Speicherplatz doppelt belegt wird

Da Bionic die LM-Studio-Runtime verwendet, kann ein korrekt in LM Studio importiertes Modell möglicherweise auch in Bionics Library erscheinen. Eine automatische gemeinsame Modellablage ist in der Bionic-Dokumentation jedoch nicht eindeutig zugesichert. Dieser Schritt muss deshalb in der installierten Version geprüft werden. `lms import` benötigt außerdem eine direkt zugängliche Modelldatei; ein Ollama-Blob sollte nicht ungeprüft wie eine gewöhnliche GGUF-Datei behandelt werden.

17.2 Muss ich die klassische LM-Studio-App installieren, um Bionic zu nutzen?

Nein, nicht grundsätzlich. Bionic ist eine eigenständige App und bringt einen eigenen Modell-Download-Manager mit (Settings > Local Models > Explore/Library, siehe Kapitel 5 und 7.1). Lokale Modelle lassen sich komplett innerhalb von Bionic herunterladen, laden und in Sessions nutzen. Die klassische App ist eine optionale Ergänzung für **Presets** (Kapitel 7.2), erweiterte technische Einstellungen und ausführlich dokumentierte lokale APIs. Klassische Presets werden dadurch aber nicht automatisch zu Bionic-Presets.

17.3 Wo werden Presets gespeichert, und muss ich sie irgendwo in Bionic „anmelden"?

Presets liegen als `.json`-Dateien lokal unter `~/.lmstudio/config-presets/` (Windows: `%USERPROFILE%\.lmstudio\config-presets`) und werden in der klassischen LM-Studio-App angelegt, bearbeitet und gelöscht. Eine Auswahl oder Registrierung dieser Presets in Bionic ist derzeit nicht offiziell dokumentiert. Die vollständige Schritt-für-Schritt-Anleitung für die klassische App steht in Kapitel 7.2.

17.4 Funktioniert Bionic auch komplett offline?

Für bereits heruntergeladene lokale Modelle im Wesentlichen ja. Ein LM-Studio-Konto ist für lokale Modelle nicht erforderlich. Internetzugriff wird unter anderem für Modell-Downloads, Updates, Cloud-Modelle und Websuche benötigt. LM Link benötigt eine Verbindung zwischen den beteiligten Geräten. Die lokale Sprachdiktierfunktion läuft nach dem erforderlichen Modelldownload auf dem Gerät.

17.5 Was ist der Unterschied zwischen einem Preset und der AGENTS.md-Regel-Datei?

Ein Preset (Kapitel 7.2) ist ein Bündel aus System Prompt und Inferenzparametern der klassischen LM-Studio-App. Eine `AGENTS.md` (Kapitel 14.1) ist dagegen eine versionierbare Projektdatei im Repository. Für Bionic ist weder die automatische Übernahme klassischer Presets noch das automatische Laden von `AGENTS.md` dokumentiert. Deshalb sollte Bionic zu Beginn einer Session ausdrücklich zum Lesen der Projektdatei aufgefordert werden.

18. Vergleich: Bionic, Claude Code, OpenCode, Ollama & Co.

> **Einordnung vorab:** Der folgende Abschnitt ist, anders als der Rest dieser Anleitung, bewusst eine persönliche, subjektive Einschätzung und keine reine Faktendarstellung. Stand ist Juli 2026, und gerade Bionic und Ollamas Agenten-Layer sind so neu, dass sich die Einschätzung in wenigen Monaten wieder verschieben kann.

Ein wichtiger Vorbehalt zuerst: Die vier genannten Werkzeuge sind keine echten Äquivalente. Bionic, Claude Code und OpenCode sind Agenten mit eigener Bedienoberfläche; Ollama ist historisch vor allem eine Modell-Laufzeit. Seit `ollama launch`, offiziell veröffentlicht im Januar 2026 und verfügbar ab Ollama 0.15, verschwimmt diese Grenze: Der Befehl konfiguriert und startet unter anderem Claude Code, OpenCode und Codex mit lokalen oder Cloud-Modellen. Ein aktuelles Beispiel ist `ollama launch claude --model qwen3.5`. Ollama gehört damit als Backend und Integrationsschicht in diesen Vergleich.

18.1 Vergleichstabelle

Werkzeug	Kategorie	Lizenz/ Offenheit	Oberfläche	Modellbindung	Reifegrad (Juli 2026)	Tool-Rechte-Modell
LM Studio Bionic	Eigenständiger GUI-Agent	Proprietär, kostenlose Basisnutzung	Desktop-App	Lokal (llama.cpp/ MLX) + eigene Cloud-Modelle	Sehr früh (wenige Tage alt)	Inline-Diffs und Checkpoints; Freigaben versionsabhängig
Claude Code	Terminal/ CLI-Agent	Proprietär	CLI + IDE-Extensions	Nativ Claude-Modelle; kompatible lokale Endpunkte wie LM Studio sind möglich	Ausgereift, lange am Markt	Granulares Permission-System (allow/deny/ask) + Hooks
OpenCode	Terminal-/ TUI-Agent mit weiteren Oberflächen	Open Source (MIT)	Terminal-TUI und weitere Clients	Provideruna bhängig, auch lokal über Ollama oder LM Studio	Etabliertes Open-Source-Projekt	Eigenes Permission-/Tool-Loop-System
Ollama	Primär Modell-Laufzeit, mit `ollama launch` auch Integrationsschicht für Agenten	Open-Source-Komponenten + optionale Cloud-Erweiterung	CLI/REST-API und Desktop-App	Lokal + optionale Cloud-Modelle	Laufzeit etabliert, Agenten-Integration neuer	Abhängig vom gestarteten Agenten, z. B. Claude Code, OpenCode oder Codex

18.2 Bionic

Bionic wirkt wie das konsequenteste „Privacy-first"-Angebot unter den vieren. Lokale Ausführung ist kein nachträglich aufgesetztes Feature, sondern die eigentliche Produktidee, inklusive lokaler Sprachsteuerung. Der Preis dafür ist unübersehbar: kein Regel-/Hook-/Skill-System (Kapitel 13), kein IDE-Plugin (Kapitel 11), und viele Settings-Bereiche sind offiziell noch kaum dokumentiert (Kapitel 7.1). Für alles, was über einfache, klar umrissene Aufgaben hinausgeht, merkt man dem Produkt sein Alter von wenigen Tagen deutlich an.

18.3 Claude Code

Claude Code gehört Stand Juli 2026 zu den technisch ausgereiftesten Werkzeugen im Vergleich, gerade wegen des Zusammenspiels aus Permission-System, Hooks, Skills und Subagenten (Kapitel 13). Nativ ist es auf Claude-Modelle und Anthropics Dienste ausgerichtet. Über kompatible Endpunkte kann es jedoch auch lokale Modelle verwenden: LM Studio dokumentiert dafür seit Version 0.4.1 den Anthropic-kompatiblen Endpunkt `/v1/messages`. Bei diesem Betrieb hängen Qualität und Werkzeugzuverlässigkeit stark vom lokalen Modell ab.

18.4 OpenCode

OpenCode ist ein providerunabhängiger Open-Source-Agent mit Tool-Loop und Session-Management, der sich sowohl mit Cloud-Anbietern als auch mit lokalen Modellen über Ollama oder LM Studio verbinden lässt. Die Ergebnisqualität hängt stark vom gewählten Modell und dessen Tool-Fähigkeiten ab. Ein schwaches lokales Modell wird durch einen anderen Agenten-Harness nicht automatisch leistungsfähig.

18.5 Ollama

Ollama ist weniger ein direkter Konkurrent als eine mögliche Infrastruktur unter anderen Agenten. Es bietet eigene APIs, darunter OpenAI- und Anthropic-kompatible Schnittstellen, und kann mit `ollama launch` Agenten wie Claude Code, OpenCode oder Codex konfigurieren und starten. LM Studio und Ollama sind dabei alternative lokale Modell-Laufzeiten; LM Studio „dockt" nicht an Ollamas API an.

18.6 Weitere nennenswerte Agenten

Es gibt deutlich mehr solcher Werkzeuge als diese vier. Ohne Anspruch auf Vollständigkeit, kurz eingeordnet:

- **Cursor** – proprietäre, ausgereifte KI-IDE auf Basis von VS Code mit starkem Fokus auf gehostete Modelle und Abonnements
- **GitHub Copilot CLI / Copilot Workspace** – tief in GitHub integriert, naheliegende Wahl für Teams, die ohnehin auf GitHub setzen
- **Aider** – schlanker, sehr Terminal-fokussierter Open-Source-Pionier unter den CLI-Coding-Agenten, weniger „Agenten-Ökosystem" (Skills, Sessions, Plan-Modi) als OpenCode, dafür minimalistisch und lange erprobt
- **Windsurf/Cascade** – IDE-Agent im Cursor-Stil, ebenfalls proprietär
- **Cline / Continue** – Open-Source-Extensions für VS Code, gute Wahl für alle, die bei ihrer bestehenden IDE bleiben wollen, statt ein separates Terminal-Tool zu lernen

18.7 Fazit

Wer eine ausgereifte, tief in ein Ökosystem integrierte Coding-Lösung sucht, sollte Claude Code prüfen. Wer Offenheit und Modellfreiheit mit einem providerunabhängigen Agenten-Harness verbinden möchte, findet in OpenCode eine interessante Alternative. Wer lokale Ausführung und eine grafische Projektoberfläche priorisiert, sollte Bionic testen, muss aber den frühen Preview-Status berücksichtigen. Ollama ist vor allem eine alternative Modell-Laufzeit und Integrationsschicht; es ist nicht das gemeinsame technische Fundament von Bionic und LM Studio.

19. Quellen

Diese Anleitung basiert auf einer Web-Recherche (Stand Juli 2026), da LM Studio Bionic ein sehr neues Produkt ist. Die wichtigsten verwendeten Quellen:

- LM Studio Bionic – offizielle Website
- Bionic-Dokumentation
- Ankündigungs-Blogpost „Introducing LM Studio Bionic"
- Bionic-Preise
- LM-Studio-App-Dokumentation
- Systemanforderungen
- Modell-Download-Anleitung
- MCP-Server in LM Studio nutzen
- Presets – offizielle LM-Studio-Dokumentation
- 9to5Mac: LM Studio launches Bionic
- Bitdoze: LM Studio Bionic Review
- GIGAZINE: LM Studio Bionic Release
- Modelle außerhalb von LM Studio importieren
- AGENTS.md – offener Standard für Coding-Agenten
- lms unload – LM Studio CLI-Dokumentation
- GitHub-Issue: Modellordner bleibt nach Löschen erhalten
- GitHub-Issue: Feature-Wunsch `lms remove`
- Ollama-Dokumentation für Windows (Modellordner)
- User- und Developer-Modus
- Server-Einstellungen
- Per-Model Defaults
- Farbschemata (Themes)
- LM Studio in verschiedenen Sprachen
- Bionic: Lokale Modelle herunterladen (Explore/Library)
- Bionic: Konten, Pläne und Abrechnung einrichten
- Bionic: Projects and Sessions
- Bionic: Code Project
- Bionic: Work Project
- Bionic: Lokale, Remote- und Cloud-Modelle
- Claude Code mit lokalen LM-Studio-Modellen
- Anthropic-kompatible API von LM Studio
- `lms import` – aktuelle CLI-Dokumentation
- OpenCode – offizielle Website
- OpenCode Developer Guide 2026
- Ollama Integrations-Dokumentation
- `ollama launch` – offizielle Ankündigung
- Ollama 2026: From Local Runner to AI Platform
- Coding and Personal Agents on Ollama (Medium)

Änderungsprotokoll

- Rund 25 der 82 Gedankenstrich-Konstruktionen im Fließtext (nicht in Überschriften, Listen-Definitionen, Tabellen oder der Quellenliste) durch Punkt, Komma oder Klammern ersetzt, um den typischen KI-Rhythmus mit gehäuften Einschüben aufzulockern
- Eine Stelle mit der abstrakten Floskel „Die Landschaft ist deutlich größer …" durch eine konkretere Formulierung ersetzt
- Struktur, Fakten, Zahlen, Tabellen, Codeblöcke und alle inhaltlichen Aussagen unverändert gelassen – der Text war bereits sachlich, gut belegt und ohne typische Werbe-/Übertreibungsfloskeln geschrieben