netidee PROJEKTE

IndeRun

Endbericht | Call 20 | Projekt ID 7913

Lizenz CC BY-SA

Inhalt

1 Einleitung 3

2 Projektbeschreibung 3

3 Verlauf der Arbeitspakete 6

3.1 Arbeitspaket 1 - Detailplanung und Formales am Projektstart 6

3.2 Arbeitspaket 2 - Architektur und Prototypen 6

3.3 Arbeitspaket 3 - Runtime-Kern mit Policy-Engine 7

3.4 Arbeitspaket 4 - LLM-Provider mit Fallback-Logik 8

3.5 Arbeitspaket 5 - Konsistente SDKs 9

3.6 Arbeitspaket 6 - Demo Apps und Auswertung 9

3.7 Arbeitspaket 7 - Dokumentation und Formales am Projektende 10

4 Umsetzung Förderauflagen 10

5 Liste Projektendergebnisse 10

6 Verwertung der Projektergebnisse in der Praxis 11

7 Öffentlichkeitsarbeit/ Vernetzung 12

8 Eigene Projektwebsite 12

9 Geplante Aktivitäten nach netidee-Projektende 13

10 Anregungen für Weiterentwicklungen durch Dritte 13

1 Einleitung

IndeRun ist eine von Independo entwickelte, offene Softwarebibliothek für die Ausführung und Steuerung von KI-Funktionen in Web-, iOS- und Android-Anwendungen. Ausgangspunkt war ein konkretes Problem unserer barrierefreien Produkte: Wir entwickeln für die Geräte, die Menschen tatsächlich verwenden („We build for the devices people actually use“), wollen aber nicht für jede Plattform eigene KI-Integrationen und Datenschutzentscheidungen pflegen. IndeRun trainiert keine eigenen KI-Modelle. Stattdessen verbindet eine gemeinsame Schnittstelle verfügbare lokale Modelle und ausdrücklich konfigurierte Cloud-Dienste; die Auswahl berücksichtigt Gerätefähigkeiten, Aufgabenanforderungen, Datenschutzregeln und erlaubte Alternativen.

Die wesentlichen technischen Projektziele wurden umgesetzt: ein gemeinsamer Routing-Kern, SDKs für TypeScript, Swift und Kotlin, Beispiele und eine schlanke Capacitor-Brücke. Für Textaufgaben stehen run() und stream() sowie vereinfachte Einstiegsfunktionen zur Verfügung; die stabile Version 0.5.0 ergänzt die Sprachausgabe. Die Softwarepakete sind über npm, Swift Package Manager und Maven Central veröffentlicht, ebenso die zugehörige Capacitor-Integration. Die Dokumentation ist unter https://inderun.dev/ öffentlich zugänglich; Quellcode und Beispiele stehen unter MIT-Lizenz auf GitHub. Eine lokale Ausführung ist stets von Gerät, Betriebssystem, installiertem Modell und Aufgabe abhängig – IndeRun verspricht keine Offline-KI auf jedem Gerät.

2 Projektbeschreibung

Ausgangslage und Projektziele

Independo entwickelt digitale Werkzeuge für Menschen, die auf Unterstützte Kommunikation angewiesen sind oder beim Lesen, Schreiben und Verstehen Unterstützung benötigen. KI kann beispielsweise Texte vereinfachen, Inhalte zusammenfassen, Formulierungen unterstützen und Informationen vorlesen. Damit solche Funktionen im Alltag zuverlässig nutzbar sind, müssen sie auf den tatsächlich verwendeten Geräten funktionieren und bei fehlender Modellverfügbarkeit verständlich reagieren.

Gleichzeitig verändert sich die KI-Landschaft sehr schnell. Apple bietet mit Foundation Models KI direkt am Gerät an, Google mit ML Kit GenAI und Gemini Nano auf Android, Browser beginnen eigene lokale Modelle bereitzustellen, und Cloud-Anbieter wie OpenAI bleiben für viele Aufgaben wichtig. Jeder dieser Wege bringt eigene Schnittstellen, Fehlerformen und Einschränkungen mit. Für kleine Entwicklungsteams bedeutet das, für jede Plattform und jeden Anbieter eigene

Sonderlogik schreiben zu müssen. Bei Anwendungen im Bereich digitaler Inklusion kommt die Frage dazu, welche Daten das Gerät überhaupt verlassen dürfen.

Ziel von IndeRun war deshalb eine offene, plattformübergreifende Ausführungs- und Routing-Schicht: Eine App formuliert die Aufgabe und ihre Datenschutzanforderungen, IndeRun prüft die konkreten Fähigkeiten des Geräts und die konfigurierten Anbieter und wählt einen zulässigen Ausführungsweg. Ergebnisse, Fehler und Herkunftsangaben werden einheitlich zurückgegeben. Ein Cloud-Fallback erfolgt nur, wenn die Anwendung ihn vorsieht, ein geeigneter Cloud-Anbieter konfiguriert ist und die Datenschutzvorgabe ihn erlaubt.

Zielgruppe

Primäre Nutzer:innen von IndeRun sind die Entwicklungsteams von Independo und andere Teams, die barrierefreie oder hybride Anwendungen für Web, iOS und Android umsetzen. Die unmittelbare Verwertung liegt in den eigenen Independo-Produkten; die MIT-Veröffentlichung macht die Technik zusätzlich für externe Entwickler:innen nutzbar. Für Endnutzer:innen sind nachvollziehbare Fehler, einheitliches Verhalten und die Möglichkeit lokaler Datenverarbeitung relevant. Offline-Funktionen sind nur dort verfügbar, wo der jeweilige lokale Anbieter und die benötigten Modelle tatsächlich einsatzbereit sind.

Inhaltlicher Überblick über das Projektergebnis

Architektur. IndeRun besteht aus drei Teilen. Der ***Engine Core*** ist das gemeinsame Herzstück und kümmert sich um Anbieter-Verwaltung, Routing-Regeln, Fallback, einheitliche Ereignisse, Abbruchlogik und Fehlerbehandlung. Die ***Host Services*** verbinden IndeRun mit der jeweiligen Plattform, etwa für Internetverbindung, Gerätezustand oder sichere Speicherung von Zugangsdaten. Die ***Provider Adapters*** kapseln die einzelnen KI-Anbieter und übersetzen deren Besonderheiten in die gemeinsame IndeRun-Sprache.

Gemeinsame Interfaces. Anfragen, Ergebnisse, Ereignisse, Fehler und Anbieterbeschreibungen sind zentral als JSON Schema definiert. Daraus werden die Datentypen für TypeScript, Swift und Kotlin erzeugt. Dadurch bedeuten Begriffe, Fehlerklassen und Ergebnisse auf allen Plattformen dasselbe.

Gemeinsame Routing-Logik. Die Entscheidung, welcher Anbieter eine Anfrage bearbeitet, wird von einem gemeinsamen Routing-Kern getroffen, der in Rust umgesetzt ist und im Web als WebAssembly läuft. Alle drei Plattformen verwenden denselben Kern. Die Entscheidung ist deterministisch: Für dieselbe Anfrage und denselben Gerätezustand entsteht immer derselbe Anbieter und dieselbe Fallback-Reihenfolge. Wird ein Anbieter ausgeschlossen, hält IndeRun fest, warum (zum Beispiel Datenschutzvorgabe, fehlende Internetverbindung oder fehlende Gerätefähigkeit).

Datenschutz als Vorgabe der Anfrage. Eine Anwendung legt pro Anfrage fest, wie streng sie ist:

- ***local_required*** (Daten dürfen das Gerät nicht verlassen, kein Cloud-Fallback),

- *local_preferred (lokale Ausführung bevorzugt; Cloud-Fallback nur bei entsprechender Konfiguration und ausdrücklicher Zulässigkeit),*
- *cloud_allowed (lokale oder ausdrücklich konfigurierte Cloud-Ausführung zulässig),*
- *cloud_required* (Daten werden in der Cloud verarbeitet)

Damit wird „On-Device oder Cloud“ zu einer prüfbaren Regel statt zu einer fest einprogrammierten Plattformentscheidung. Unter local_required muss IndeRun eine Anfrage ablehnen, wenn kein geeigneter lokaler Anbieter verfügbar ist; es darf dann nicht stillschweigend in die Cloud wechseln. Ein nicht verfügbarer Anbieter wird mit nachvollziehbaren Gründen ausgewiesen. Cloud-Zugangsdaten werden nicht als Klartext in die Aufgabe geschrieben, sondern über die vorgesehene sichere Konfiguration eingebunden.

Abbildung 1 zeigt, wie eine Anfrage durch IndeRun läuft.



Abbildung 1: Ablauf einer KI-Anfrage in IndeRun – von der einheitlichen Anfrage über die Prüfung von Gerät, Verbindung, Zugangsdaten und Datenschutzvorgabe bis zum einheitlichen Ergebnis.

Capability-Checks. Ob ein lokales Modell tatsächlich genutzt werden kann, hängt vom Betriebssystem, vom konkreten Gerät, von der Verfügbarkeit des Modells und von der Verbindung ab. IndeRun fragt diese Fähigkeiten zur Laufzeit ab. Über ***checkCapabilities()*** kann eine App außerdem vorab prüfen, welche Möglichkeiten auf dem aktuellen Gerät verfügbar sind, und ihre Benutzeroberfläche danach ausrichten.

Unterstützte Anbieter für Textaufgaben. Zum Projektende sind folgende Ausführungswege integriert; ihre tatsächliche Nutzbarkeit wird pro Gerät und Aufgabe geprüft:

- ein OpenAI-kompatibler Cloud-Anbieter auf allen Plattformen (auch nutzbar mit OpenRouter),
- Apple Foundation Models auf iOS und macOS,
- Android ML Kit GenAI (Gemini Nano),
- ONNX Runtime für eigene, von Entwickler:innen mitgelieferte Modelle auf Web, iOS und Android,
- die Prompt API des Browsers (Chrome) als browserseitiges lokales Modell.

Ausführungsarten und API. IndeRun unterstützt run() für abgeschlossene Textanfragen und stream() für schrittweise Ergebnisse mit Abbruchmöglichkeit. generateText() und streamText() erleichtern den Einstieg. Modellprofile begrenzen zulässige Ausführungswege; Ergebnisse enthalten Angaben zum genutzten Anbieter und zur Ausführungsumgebung. Mit Version 0.5.0 kamen außerdem die Sprachausgabe-Funktionen speak(), generateSpeech() und listVoices() hinzu. Sie verwenden je nach Plattform verfügbare Systemstimmen; die Eigenschaften der Stimmen und die Frage, ob sie ohne Netz verfügbar sind, unterscheiden sich nach Gerät und Betriebssystem.

Plattformen und Pakete. Das Web-SDK @independo/inderun-web wird über npm, das Swift Package IndeRun über das GitHub-Repository und die Android-Bibliothek app.independo.inderun:inderun-kotlin über Maven Central bereitgestellt. Die ebenfalls veröffentlichte Capacitor-Integration @independo/capacitor-inderun verbindet hybride Anwendungen mit den nativen SDKs; Routing und Fallback verbleiben in der Runtime. Ein zentraler Einstieg mit Schnellstart, Architektur- und Kompatibilitätsinformationen ist unter https://inderun.dev/ erreichbar.

Veröffentlichungsstand und Grenzen. IndeRun 0.5.0 und die zugehörige Capacitor-Integration sind veröffentlicht. Neben Text-zu-Text-Aufgaben ist die systembasierte Sprachausgabe integriert; speak(), generateSpeech() und listVoices() stehen für die unterstützten Plattformen zur Verfügung. Zusätzliche neuronale On-Device-TTS-Modelle wie Articulino wurden auf realen Geräten experimentell überprüft, sind aber nicht als allgemein freigegebene Produktionsmodelle ausgewiesen. Browser, iOS und Android unterscheiden sich bei Stimmverfügbarkeit, lokaler Ausführung und Audio-Artefakten. Bidirektionale Echtzeit-Sitzungen („Mode 3“) bleiben eine geplante Weiterentwicklung.

3 Verlauf der Arbeitspakete

3.1 Arbeitspaket 1 - Detailplanung und Formales am Projektstart

Kurzbeschreibung der Haupttätigkeiten: Die Haupttätigkeiten in Arbeitspaket 1 umfassen die Vertragsprüfung und -unterzeichnung. Weiter wurde die Projektwebseite überprüft, mit Details zum Team ergänzt und das Abrufformular für die erste Förderrate abgeschickt. Nachdem die Webseite angepasst wurde, begann die Detailplanung des Projekts IndeRun, es wurden Arbeitspakete für die Core-Library, Mobile- und Backend-Entwicklung geschnürt.

Besondere Erfolge/Probleme: Bei der Detailplanung und dem Projektstart traten keine nennenswerten Probleme auf.

Abweichungen zum Plan: Keine.

Erkenntnisse zur Vorgangsweise: Die frühe Aufteilung in klar abgegrenzte Arbeitspakete hat die spätere Umsetzung erleichtert.

3.2 Arbeitspaket 2 - Architektur und Prototypen

Kurzbeschreibung der Haupttätigkeiten: Ziel dieses Arbeitspakets war eine einheitliche Architektur und ein erstes Proof-of-Concept für den Provider-Flow. Dazu wurden das Repository, die Build-Pipelines und das Tooling eingerichtet, die Architektur erarbeitet und dokumentiert sowie ein erster durchgängiger Ablauf von der App-Anfrage bis zum Modell umgesetzt.

In diesem Arbeitspaket wurde geklärt, wie IndeRun grundsätzlich funktionieren soll: App-Entwickler:innen sollen nicht für jeden KI-Anbieter eigene Integrationen schreiben müssen. Stattdessen nimmt IndeRun eine Anfrage entgegen, prüft die verfügbaren Möglichkeiten und leitet die Anfrage an einen passenden Anbieter weiter. Festgelegt wurden die Aufteilung in ***Engine Core***, ***Host Services*** und ***Provider Adapters***, die drei API-Grundformen (*Run*, *Stream*, *Session*) sowie JSON Schema als gemeinsame Quelle für alle Datenformate. Die Architekturentscheidungen sind im Repository als Architekturdokumentation und Entscheidungsprotokolle (ADRs) festgehalten.

Besondere Erfolge/Probleme: Aus dem Architekturkonzept sind früh erste lauffähige Bausteine für Web, iOS und Android entstanden.

Abweichungen zum Plan: Keine wesentlichen Abweichungen.

Erkenntnisse zur Vorgangsweise: Die bewusste Entscheidung „stabile Architektur vor möglichst langer Anbieterliste“ hat sich bewährt. Weil Interfaces, Fehlerformen und Routing-Entscheidungen von Anfang an sauber definiert waren, konnten später neue Anbieter und Ausführungsarten ergänzt werden, ohne die Grundlagen neu zu bauen.

3.3 Arbeitspaket 3 - Runtime-Kern mit Policy-Engine

Kurzbeschreibung der Haupttätigkeiten: Die Haupttätigkeit dieses Arbeitspakets bestand in der Implementierung der zentralen Runtime mit Provider-Registry, Policy-/Routing-Regeln, Fehlerbehandlung, Task-Schemas und Capability-Checks.

Die zentrale Bibliothek nimmt Anfragen entgegen, wählt Anbieter nach Regeln aus und gibt Ergebnisse oder Fehler in einer einheitlichen Form zurück. Anwendungen müssen dabei nicht wissen, welcher Anbieter im Hintergrund genutzt wird. Seit dem Zwischenbericht wurden insbesondere folgende Punkte umgesetzt: die Laufzeitabfrage der Gerätefähigkeiten (***checkCapabilities()***), die Ausführungsart (*Run* oder *Stream*) als Teil der Routing-Entscheidung, Modellprofile zur Steuerung erlaubter Modelle sowie Herkunftsangaben (Provenance) in jedem Ergebnis.

Besondere Erfolge/Probleme: Ein wichtiger Schritt war die Vereinheitlichung der Routing-Logik. Anfangs gab es neben dem gemeinsamen Routing-Kern noch plattformspezifische Nachbauten derselben Regeln in TypeScript, Swift und Kotlin. Diese wurden entfernt; seit Version 0.3 entscheidet derselbe in Rust umgesetzte Routing-Kern auf allen Plattformen, wohin eine Anfrage geroutet wird. Gemeinsame Testfälle werden von den TypeScript-, Swift- und Kotlin-Testsuiten geladen und prüfen das übereinstimmende Verhalten.

Abweichungen zum Plan: Keine wesentlichen Abweichungen.

Erkenntnisse zur Vorgangsweise: „On-Device unterstützt“ allein reicht nicht aus. Ob ein Anbieter tatsächlich genutzt werden kann, ist eine Laufzeitentscheidung. Ebenso wichtig wie die Entscheidung selbst ist, nachvollziehbar zu machen, warum ein Anbieter ausgewählt oder ausgeschlossen wurde; gerade beim Testen auf unterschiedlichen Geräten ist diese Transparenz entscheidend.

3.4 Arbeitspaket 4 - LLM-Provider mit Fallback-Logik

Kurzbeschreibung der Haupttätigkeiten: Im Arbeitspaket 4 lag der Fokus auf der Integration der wichtigsten LLM-Provider (Anbieter für große Sprachmodelle) und auf dem Fallback-Verhalten zwischen lokalen und Cloud-Anbietern.

Integriert sind ein *OpenAI-kompatibler Cloud-Anbieter* (plattformübergreifend, auch für OpenRouter nutzbar), *Apple Foundation Models* auf iOS/macOS und *Android ML Kit GenAI* (Gemini Nano). Über den ursprünglichen Plan hinaus wurden eine *ONNX-Runtime* für eigene lokale Modelle auf Web, iOS und Android inklusive eines gemeinsamen Modell-Paketformats sowie ein browserseitiger lokaler Anbieter auf Basis der Chrome Prompt API umgesetzt. Die Fallback-Logik schließt Anbieter, die eine Aufgabe nicht erfüllen können, bereits vor der Ausführung aus und versucht anschließend die verbleibenden Anbieter in einer festen Reihenfolge. Die Anbieterübersicht im Repository zeigt, welcher Anbieter auf welcher Plattform verfügbar ist, welche Ausführungsarten er unterstützt und welche Grenzen er hat.

Besondere Erfolge/Probleme: Lokale KI ist von Hardware, Betriebssystemfunktionen, Modellverfügbarkeit und unterstützter Aufgabe abhängig. Apple Foundation Models benötigen geeignete Apple-Intelligence-Geräte, Android ML Kit GenAI setzt entsprechende AICore-/Gemini-Nano-Unterstützung voraus. Nicht alle lokalen Anbieter unterstützen Streaming; eine fehlende Fähigkeit wird vor der Ausführung erkannt. Die Integration von Sprachausgabe hat zusätzlich gezeigt, dass lokale oder netzbasierte Stimmen je nach Betriebssystem und konkreter Installation unterschiedlich verfügbar sind.

Abweichungen zum Plan: Der optional vorgesehene Hugging-Face-Anbieter wurde nicht als eigener Cloud-Provider umgesetzt. Stattdessen wurde die ONNX-Runtime für eigene, von Anwendungen bereitgestellte Modelle auf mehreren Plattformen integriert. Die Inferenz kann damit lokal erfolgen, sofern das Modell auf dem Gerät vorhanden, kompatibel und einsatzbereit ist.

Erkenntnisse zur Vorgangsweise: Fallback bedeutet nicht einfach „Fehler abfangen und den nächsten Anbieter probieren“. Erst wenn vorab geprüft wird, welche Anbieter Aufgabe, Ausführungsart und Datenschutzvorgabe überhaupt erfüllen können, entsteht ein Verhalten, das für Anwendungen vorhersehbar ist.

3.5 Arbeitspaket 5 - Konsistente SDKs

Kurzbeschreibung der Haupttätigkeiten: Entwickelt wurden konsistente TypeScript-, Swift- und Kotlin-SDKs mit gemeinsam beschriebenen Schnittstellen, Packaging, Versionierung und automatisierten Release-Prüfungen. Zur Text-API kamen in Version 0.5.0 die plattformübergreifenden Schnittstellen für Sprachausgabe hinzu.

Die öffentliche Schnittstelle (`IndeRunApi`) wird aus den gemeinsamen Interfaces für alle drei Sprachen erzeugt, sodass Benennungen, Fehlerbehandlung und Abläufe übereinstimmen. Mit Version 0.4 kamen die einfachen Einstiegsfunktionen ***generateText()*** und ***streamText()*** dazu; die Schichtung zwischen einfacher und fortgeschrittener API sowie die Kompatibilitätsregeln sind dokumentiert. Das Web-SDK wurde in einen Einstiegspunkt für Anwendungen und einen für die Entwicklung eigener Anbieter getrennt.

Die Kernpakete wurden über npm, Maven Central und Swift Package Manager veröffentlicht; die stabile Version 0.5.0 umfasst auch die systembasierte Sprachausgabe. Automatische Prüf- und Veröffentlichungsprozesse auf GitHub Actions bauen und testen JavaScript, Rust, Swift und Android und erzeugen versionierte Releases mit Änderungsprotokollen. Die Capacitor-Brücke wird in einem eigenen Repository entwickelt und ist ebenfalls veröffentlicht. Damit können auch hybride Apps die nativen IndeRun-Funktionen verwenden, ohne eine zusätzliche Routing-Implementierung zu benötigen.

Besondere Erfolge/Probleme: Keine besonderen Vorkommnisse.

Abweichungen zum Plan: Keine besonderen Vorkommnisse.

Erkenntnisse zur Vorgangsweise: Eine einzige Quelle für alle Datenformate und eine aus ihr erzeugte API verhindern, dass drei Implementierungen derselben Logik auseinanderlaufen. Dasselbe gilt für die Capacitor-Brücke: Sie bewusst „dünn“ zu halten, verhindert eine vierte, leicht abweichende Implementierung.

3.6 Arbeitspaket 6 - Demo Apps und Auswertung

Kurzbeschreibung der Haupttätigkeiten: In Arbeitspaket 6 wurden Demo-Apps für Web, iOS und Android, Pilot-Integrationen in zwei Independo-Anwendungen sowie technische und qualitative Verifikationen erarbeitet. Dazu gehörten praktische Tests der Sprachausgabe und Benchmarks auf realer iOS- und Android-Hardware.

Für alle drei Plattformen stehen Demo-Apps zur Verfügung. Die Web-Demo zeigt anhand der Vereinfachung schwer verständlicher Texte, wie eine Anwendung Datenschutzvorgaben für lokale oder erlaubte Cloud-Verarbeitung vorgibt und den tatsächlich gewählten Anbieter sichtbar macht. Die nativen Demos zeigen Capability-Checks, Anbieterauswahl und Streaming; die Sprach-Demos und Testwerkzeuge ergänzen dies um Stimmabfrage, Wiedergabe, Abbruch und Audio-Erzeugung. Fehlende Anbieter oder Funktionen werden als solche angezeigt und nicht als erfolgreiche Offline-Ausführung ausgegeben.

IndeRun wurde in zwei Independo-Anwendungen pilotiert: Im Independo Portal wird eine kurze Tageszusammenfassung erzeugt; im Independo Kalender wird die Vorlesefunktion für Kalendereinträge erprobt. Beide Integrationen befinden sich noch in der Testphase und sind nicht als abgeschlossene, flächendeckende Produkteinführung zu verstehen.

Das Feedback aus der Testgruppe war für belastbare Aussagen zur Wirkung in den Pilot-Anwendungen noch zu klein. Davon getrennt wurden konkrete Sprachsynthese-Tests und technische Messungen auf realen Geräten durchgeführt.

Technische Verifikation: Am 9. und 10. November 2026 wurde das neuronale Articulino-TTS-Modell (ONNX int8, CPU-Ausführung) auf einem iPhone 15 und einem Google Pixel 9 Pro gemessen. Für einen kurzen deutschen Text betrug die mediane warme Renderzeit 71 ms auf dem iPhone und 887 ms auf dem Pixel; für einen langen Text 1.023 ms bzw. 10.719 ms. Die entsprechenden Echtzeitfaktoren (Renderzeit im Verhältnis zur Audiodauer) lagen bei 0,030 bzw. 0,377 für den kurzen und 0,028 bzw. 0,298 für den langen Text. Die Android-Tests enthielten zusätzlich einen Vergleich zur dortigen System-Sprachausgabe. Die Messungen stammen jeweils von einem Gerät und einer kurzen Testserie; sie sind keine allgemeine Aussage zu allen Geräten oder zur Akkulaufzeit.

Hörbeurteilung und Einordnung: Die erzeugte Sprache wurde vom Team auf beiden Geräten angehört. Klangcharakter und wahrgenommene Qualität unterschieden sich je nach gewählter Articulino-Stimme und der zugrunde liegenden Referenz-Sprachprobe; alle getesteten Stimmen wurden auf beiden Geräten subjektiv als gut beurteilt. Diese praktische Hörbeurteilung ist keine standardisierte Studie mit quantitativen Hörwerten. Die neuronalen Modellpakete sind derzeit als

benchmark_only dokumentiert; eine rechtliche Freigabe für den Produktiveinsatz steht noch aus. Messmethode, Rohdaten und Einschränkungen sind öffentlich dokumentiert: https://github.com/independo-gmbh/inderun/tree/dev/docs/verification (insbesondere neural-speech.md, neural-speech-admission.md und reports/).

Besondere Erfolge/Probleme: Keine besonderen Vorkommnisse.

Abweichungen zum Plan: Keine Abweichungen zum Plan.

Erkenntnisse zur Vorgangsweise: Demos und Prüfungen sollten nicht nur erfolgreiche Ergebnisse zeigen, sondern ebenso nicht verfügbare lokale Anbieter, datenschutzbedingte Ablehnungen, Abbruchverhalten und die Grenzen einzelner Geräte. Reale Geräte-Benchmarks ergänzen automatisierte Tests; Laufzeit, Speicherbedarf und subjektive Audioqualität müssen getrennt beurteilt werden. Messwerte von einzelnen Geräten sind nicht ohne Weiteres auf alle iOS- oder Android-Geräte übertragbar.

3.7 Arbeitspaket 7 - Dokumentation und Formales am Projektende

Kurzbeschreibung der Haupttätigkeiten: Arbeitspaket 7 umfasst den Projektendbericht, die veröffentlichungsfähige Zusammenfassung, die Endabrechnung, die Dokumentation der Ergebnisse und die formalen Einreichungen bei netidee. Die technischen Ergebnisse und Quelltexte sind auf GitHub und unter https://inderun.dev/ dokumentiert. Endbericht, Einseiter, Abrechnung und gegebenenfalls noch ausstehende Uploads beziehungsweise Fördermittelabrufe werden im Rahmen der Abschlussabwicklung nach den netidee-Vorgaben eingereicht; mit diesem Bericht allein wird deren gesonderte Durchführung nicht als bereits erledigt behauptet.

Die öffentlich zugängliche Dokumentation umfasst Schnellstarts, API und Plattformhinweise unter https://inderun.dev/ sowie im GitHub-Repository Architektur, Schnittstellen, Streaming, ONNX-Runtime, CI, Release-Prozesse und Migrationsanleitungen. Die Geräteverifikation ist mit datierten Benchmarks und maschinenlesbaren Rohberichten unter https://github.com/independo-gmbh/inderun/tree/dev/docs/verification nachvollziehbar. Die zusätzlich erforderlichen Projektdokumente werden auf der netidee-Projektseite https://www.netidee.at/inderun bereitgestellt.

4 Umsetzung Förderauflagen

In der Fördervereinbarung wurden keine zusätzlichen projektspezifischen Sonderauflagen festgelegt. Die allgemeinen Förderbedingungen bleiben davon unberührt: insbesondere die Veröffentlichung der Projektergebnisse unter geeigneten offenen Lizenzen, die geforderten Berichte und Dokumentationen, die Kennzeichnung der netidee-Förderung sowie die Abschlussabrechnung und Bereitstellung der vorgesehenen Inhalte über die netidee-Projektseite.

5 Liste Projektendergebnisse

Nr.	Projektendergebnis	Lizenz	Ort
1	**Projektzwischenbericht**	CC BY-SA 4.0	https://www.netidee.at/inderun
2	**Projektendbericht**	CC BY-SA 4.0	https://www.netidee.at/inderun
3	**Entwickler:innen-Dokumentation: Systemkonzept, Architektur und Provider-Schnittstellen, API-Schichtung, Streaming, Release- und Migrationshinweise sowie dokumentierte Test- und Verifikationsverfahren für Sprachausgabe**	CC BY-SA 4.0	https://www.netidee.at/inderun https://github.com/independo-gmbh/inderun https://inderun.dev/docs
4	**Anwender:innen- und Einstiegsdokumentation: Was ist IndeRun, Schnellstart für Web, iOS, Android und Capacitor, Konfiguration, Datenschutzvorgaben, Capability-Checks und Grenzen lokaler Ausführung**	CC BY-SA 4.0	https://www.netidee.at/inderun https://github.com/independo-gmbh/inderun https://inderun.dev/docs
5	**Veröffentlichungsfähiger Einseiter / Zusammenfassung**	CC BY-SA 4.0	https://www.netidee.at/inderun
6	**Dokumentation Externkommunikation** (als Teil dieses Endberichts, Kapitel 7)	CC BY-SA 4.0	https://www.netidee.at/inderun
7	**IndeRun Runtime und Cross-Platform-SDKs: gemeinsamer Routing-Kern (Rust/WebAssembly), regelbasierte Anbieterauswahl mit kontrolliertem Fallback, Capability-Checks, Textanbieter (OpenAI-kompatibel, Apple Foundation Models, ML Kit GenAI, ONNX Runtime, Chrome Prompt API), Sprachausgabe-API mit Systemstimmen sowie TypeScript-, Swift- und Kotlin-SDKs; Kernversion 0.5.0 auf npm, Maven Central und über Swift Package Manager veröffentlicht**	MIT	https://github.com/independo-gmbh/inderun

Nr.	Projektendergebnis	Lizenz	Ort
8	**IndeRun Demo-Apps und Referenz-Integrationen: Demos für Web, iOS und Android zu Anbieterauswahl, Datenschutzvorgaben, Fallback, Streaming und Sprachausgabe; ergänzt um reproduzierbare Testwerkzeuge und reale Geräte-Benchmarks**	MIT	https://github.com/independo-gmbh/inderun
9	**Veröffentlichte Capacitor-Integration @indepedo/capacitor-inderun: schlanke Brücke für hybride Apps zu den Web-, iOS- und Android-SDKs, einschließlich der integrierten Sprachausgabe**	MIT	https://github.com/independo-gmbh/capacitor-inderun

6 Verwertung der Projektergebnisse in der Praxis

IndeRun ist ein technischer Baustein für die barrierefreien Anwendungen von Independo und wird nach dem Förderprojekt innerhalb der eigenen Produkte weitergepflegt. Aktuell befinden sich zwei Integrationen in Erprobung: Im Independo Portal entsteht eine Tageszusammenfassung, im Independo Kalender wird die Vorlesefunktion für Termine getestet. Die Pilotphase dient dazu, Verfügbarkeit lokaler Modelle, nutzerverständliche Fehler und gegebenenfalls erlaubte Alternativen unter praktischen Bedingungen zu überprüfen; ein allgemeiner Produktivrollout ist damit noch nicht belegt.

Die unter MIT-Lizenz veröffentlichten Softwarepakete können auch von externen Teams unter Einhaltung der Lizenzbedingungen in eigene Anwendungen eingebunden und erweitert werden. Die öffentliche Dokumentation unter https://inderun.dev/ erleichtert diesen Zugang. Ein eigenständiges kommerzielles Geschäftsmodell für IndeRun ist nicht geplant: Der strategische Nutzen liegt in einer gemeinsamen, wartbaren KI-Ausführungsschicht für die Independo-Produkte und in den offen veröffentlichten Ergebnissen der netidee-Förderung. Externe Nutzung und Beiträge sind willkommen, aber keine Voraussetzung für die Verwertung. Nach bisherigem Stand sind noch keine externen Beiträge dokumentiert.

7 Öffentlichkeitsarbeit/ Vernetzung

Der Projektleiter hat am netidee Summer Talk teilgenommen und sich mit den Teilnehmer:innen der anderen, im Call 20 geförderten Projekte ausgetauscht.

Auf der Projektwebseite wurden vier Berichte zum Projektfortschritt veröffentlicht:

- „IndeRun – Kick-Off“

- „IndeRun: Warum wir zuerst die Architektur klären“
- „IndeRun v0.1 ist veröffentlicht“
- „On-Device KI mit Cloud-Fallback: IndeRun wird praktisch nutzbar“

Die Entwicklung findet öffentlich auf GitHub statt. Releases werden mit Versionsständen und Änderungsprotokollen nachvollziehbar dokumentiert; Architekturentscheidungen und anstehende Aufgaben sind im Repository einsehbar. Im Beitrag zu Version 0.1 wurde die Entwickler:innen-Community um Rückmeldungen zu Architektur, Capability-Checks und Fehlerbehandlung gebeten. Mit der stabilen Version 0.5.0 sowie der veröffentlichten Capacitor-Integration liegen weitere überprüfbare Software-Ergebnisse vor. https://inderun.dev/ ist als öffentliche Entwickler:innen-Website mit Einstieg, Installation, Architektur und Plattformhinweisen online; zusätzlich wird IndeRun unter https://www.independo.app/open-source vorgestellt.

8 Eigene Projektwebsite

Seit Projektabschlussvorbereitung ist mit https://inderun.dev/ eine öffentlich erreichbare, eigenständige Entwickler:innen-Website verfügbar. Sie führt vom Anwendungsfall über Schnellstart und Plattformanforderungen zur technischen Dokumentation und macht die Grenzen lokaler Ausführung nachvollziehbar. Ergänzend stehen die Quelltexte und Releases in zwei GitHub-Repositories:

- IndeRun GitHub Repo
- Capacitor-Integration GitHub Repo

Weitere Projektinformationen sind auf der Independo-Open-Source-Seite zu finden. Die netidee-Projektseite https://www.netidee.at/inderun bleibt der Ort für die förderbezogenen Berichte und Veröffentlichungen; die eigene Entwickler:innen-Website ersetzt diese Einreichungen nicht:

- Independo Open Source

9 Geplante Aktivitäten nach netidee-Projektende

Wartung, Sicherheitspflege, technische Dokumentation und die Weiterentwicklung von IndeRun werden nach Projektende von Independo übernommen, damit die gemeinsame Runtime in den eigenen barrierefreien Anwendungen nutzbar bleibt. Die MIT-Lizenz ermöglicht daneben externe Nutzung und Mitarbeit, ohne dass ein neues eigenständiges Geschäftsmodell aufgebaut werden muss. Folgende Weiterentwicklungen sind vorgesehen:

- Verbesserung der bereits veröffentlichten Entwickler:innen-Website https://inderun.dev/ mit zusätzlichen reproduzierbaren Beispielen, einer deutlicheren Kompatibilitätsübersicht und Demonstrationen für zulässige lokale und Cloud-Ausführungswege.
- Qualitätssicherung der in Version 0.5.0 veröffentlichten Sprachausgabe auf weiteren realen Geräten, zusätzliche Hör- und Kompatibilitätstests sowie Prüfung der noch offenen technischen und rechtlichen Freigaben für neuronale TTS-Modelle. Deren Benchmark-Ergebnisse bedeuten noch keine allgemeine Produktionsfreigabe.
- Streaming auch für die ONNX-Runtime- und die browserseitigen lokalen Anbieter.
- Echtzeit-Sitzungen (Mode 3) für bidirektionale Interaktionen.
- Fortsetzung der Pilot-Tests in den barrierefreien Independo-Anwendungen, insbesondere die Überprüfung und mögliche produktive Übernahme der Funktionen im Portal und im Kalender nach Abschluss der Erprobung.

10 Anregungen für Weiterentwicklungen durch Dritte

IndeRun ist so aufgebaut, dass andere Teams zusätzliche Anbieter und Integrationen entwickeln können. Dafür sind eine Provider-Schnittstelle und gemeinsame Testfälle dokumentiert. Die folgenden Ideen sind mögliche Weiterentwicklungen und nicht Bestandteil der bereits ausgelieferten Projektergebnisse:

- **Weitere Anbieter:** etwa lokale Laufzeiten wie llama.cpp, MediaPipe LLM Inference, Core ML oder ExecuTorch sowie – getrennt davon und nur bei zulässiger Cloud-Verarbeitung – zusätzliche externe Dienste wie die Hugging Face Inference API.
- **Weitere Aufgabentypen:** Die systembasierte Sprachausgabe ist seit Version 0.5.0 bereits umgesetzt. Künftige Erweiterungen könnten beispielsweise Spracherkennung, Bildbeschreibung oder Bildklassifikation betreffen.

- **Weitere Brücken für hybride Frameworks**: Nach dem Vorbild der schlanken Capacitor-Integration können Brücken für andere hybride Frameworks gebaut werden.
- **Eigene Anwendungen:** Insbesondere in Barrierefreiheit und digitaler Inklusion – etwa für einfache Sprache, Unterstützte Kommunikation und Vorlesefunktionen – kann IndeRun die Integration geeigneter KI-Funktionen vereinfachen. Ein konkreter lokaler oder offlinefähiger Einsatz bleibt von den Fähigkeiten des jeweiligen Geräts und den verfügbaren Modellen abhängig.