ChatGPT

Planung und Tech-Stack für ein KI-gestütztes Sprachdokumentations-Tool in der Zahnarztpraxis

Projektziele und Anforderungen

Das Ziel des Projekts ist es, ein **Sprachassistenz-Tool für Zahnärzte** zu entwickeln, mit dem der Zahnarzt und seine Assistenz die Behandlungsdokumentation **per Spracheingabe** erfassen können. Die erfassten Befunde und Behandlungsschritte sollen automatisch in **abrechnungsrelevante Positionen** (z.B. BEMA/GOZ-Ziffern) umgewandelt und in die Praxissoftware *Evident* übernommen werden. Auf diese Weise entfällt das manuelle Tippen, und alle erbrachten Leistungen werden vollständig dokumentiert und abgerechnet. Ähnliche KI-Lösungen wie die *Doctos*-App für EVIDENT zeigen bereits, dass ein solches Vorgehen möglich ist: Dabei wird die Sprache zunächst in Text transkribiert und dann von KI **automatisch nach abrechnungsfähigen Ziffern durchsucht** [1]. Der Fokus liegt darauf, *alle* praxisrelevanten Dokumentationsbereiche abzudecken – **Befunddokumentation, Leistungsdokumentation (Abrechnung) und Behandlungsplanung** –, und dies möglichst **in Echtzeit während der Behandlung** (ohne dass eine Assistenz mitschreiben muss).

Wichtig ist, dass die Lösung **praktikabel in den Praxisalltag** integrierbar ist. In der Praxis des Fragestellers wird die Verwaltungssoftware *Evident* lokal betrieben, daher muss die entwickelte Anwendung mit dieser Software kompatibel sein (möglichst über eine geeignete Schnittstelle oder Datenübertragung in Evident). Als Eingabegerät schwebt ein **Tablet (z.B. iPad)** vor, das der Behandler während der Behandlung leicht nutzen kann – ggf. per Halterung oder durch die Assistenz bedient. Falls die Entwicklung einer nativen iPad-App zu Beginn zu aufwendig ist, kann zunächst eine **Webanwendung** entwickelt werden, die im Browser des iPads läuft.

Nutzen und Motivation: Ein solches System soll Zeit sparen und die Dokumentationsqualität erhöhen. Durch sprachgesteuerte Dokumentation kann pro Tag erheblich Zeit gewonnen werden, die stattdessen für Patienten genutzt wird (vergleichbare Lösungen berichten von **über 1 Stunde Zeitersparnis täglich** [2]). Zudem verhindert die automatische Erkennung von Leistungen, dass etwas **vergessen wird** – was laut Erfahrungen bis zu *15% Umsatzsteigerung* bringen kann, weil alle erbrachten Leistungen tatsächlich abgerechnet werden [3]. Insgesamt soll das Tool den **Dokumentationsaufwand senken** und die **Abrechnung optimieren**, sodass das Praxisteam entlastet wird und keine wichtigen Einträge übersehen werden.

Systemkonzept und Architekturüberblick

Um diese Ziele zu erreichen, muss das System mehrere Komponenten nahtlos verbinden. Im Kern sieht die **Architektur** folgendermaßen aus:

1. **Frontend (Benutzerschnittstelle)** – Ein Interface auf dem iPad (oder PC-Webbrowser), das es ermöglicht, Sprachaufnahmen zu starten/stoppen und ggf. Ergebnisse anzuzeigen. Dieses Frontend kann zunächst als Web-App (HTML/JS/CSS) umgesetzt werden, was schnelles

Prototyping ermöglicht [4]. Später kann eine native iPad-App entwickelt werden, wenn nötig, um z.B. Offline-Fähigkeit oder bessere Mikrofon-Nutzung zu erreichen.

2. **Sprachaufnahme & -Übertragung** – Die gesprochene Sprache des Zahnarztes wird per Mikrofon aufgenommen. In einer Webanwendung kann hierzu die Web Speech API oder ein eigenes Aufnahmeskript genutzt werden. Die Audio-Daten werden dann an einen Backend-Server gesendet (idealerweise gestreamt, um nahezu in Echtzeit transkribieren zu können).

3. **Speech-to-Text Engine (ASR)** – Ein zentrales Modul wandelt das Audio in geschriebenen Text um. Hier kommen **Speech-to-Text-Algorithmen** bzw. Modelle zum Einsatz, die medizinisches/deutsches Vokabular gut verstehen. Optionen sind z.B. das **OpenAI-Whisper**-Modell (open-source und relativ genau, kann lokal ausgeführt werden) oder Cloud-Dienste wie Google Cloud Speech, Amazon Transcribe, Microsoft Azure Speech. Für den Prototyp kann ein Cloud-STT-Service genutzt werden, um schnell Ergebnisse zu bekommen. Langfristig – auch aus **Datenschutzgründen** – wäre ein lokal laufendes Modell (oder ein Cloud-Dienst mit Gesundheitsdaten-Zertifizierung) sinnvoll. Die Genauigkeit der Transkription ist entscheidend, da Fehler hier zu falschen Vorschlägen führen könnten. Medizinische Fachbegriffe (z.B. Zahnnamen, lateinische Begriffe, Abkürzungen) sollten dem System beigebracht oder als **Custom Vocabulary** hinterlegt werden, damit die Erkennung präzise ist [5]. Moderne medizinische Spracherkennung kann sogar Sprechertrennung (Arzt vs. Patient) vornehmen [5] – für unsere Anwendung ist hauptsächlich der Arzt relevant, aber falls Hintergrundgespräche mitlaufen, könnte eine Filterung nützlich sein.

4. **NLP/AI-Modul für Inhaltserkennung und Kodierung** – Der transkribierte Text wird anschließend von einer **KI** analysiert. Dieses Modul – wahrscheinlich das **Herzstück** des Projekts – muss erkennen, **welche klinischen Leistungen, Diagnosen oder Planungen im Text erwähnt wurden**, um daraus strukturierte Einträge zu erzeugen. Hier kommen **Large Language Models (LLMs)** oder spezialisierte NLP-Modelle ins Spiel. Die KI sollte zum einen eine **strukturierte Dokumentation** erstellen (vergleichbar mit den klassischen Karteikarteneinträgen oder SOAP-Notizen aus dem Text) und zum anderen die relevanten **Abrechnungsziffern** identifizieren. Bereits vorhandene Lösungen zeigen, dass LLMs in der Lage sind, Freitext in strukturierte medizinische Dokumente und Codes zu überführen [6] [7]. Beispielsweise nutzen einige kommerzielle Coding-Lösungen KI/LLM, um unstrukturierte medizinische Notizen in ICD/CPT-Codes zu überführen [7] – analog soll unsere KI gesprochene zahnärztliche Befunde in BEMA/GOZ-Positionen umwandeln.

Wie könnte das KI-Modul konkret arbeiten? Ein Ansatz wäre, einen vortrainierten **LLM (wie GPT-4)** per **Prompt-Engineering** so zu steuern, dass es aus dem Transkript die gewünschten Informationen extrahiert. Man könnte z.B. einen Prompt entwerfen: *"Lies den folgenden Behandlungstext und extrahiere: 1) relevante Befunde/Diagnosen, 2) durchgeführte Leistungen mit passenden GOZ/BEMA Ziffern, 3) empfohlene nächste Schritte (Behandlungsplan). Gib das Ergebnis in strukturierter Form aus."* Das LLM würde dann einen strukturierten Vorschlag liefern. Alternativ oder ergänzend kann man regelbasierte Ansätze nutzen: Z.B. eine Liste von Schlüsselwörtern oder Phrasen, die bestimmten Abrechnungspositionen zugeordnet sind, um die KI-Ausgabe abzusichern. In Version 1 kann es genügen, wenn die KI **Vorschläge** für Abrechnungspositionen macht, die der Arzt bestätigt. Die *Doctos*-App in EVIDENT geht in ihrer neuesten Version ähnlich vor – sie **durchsucht relevante Gebührenkataloge (BEMA, GOZ, GOÄ) automatisch** nach passenden Positionen zum gesprochenen Text und schlägt diese vor [8]. Sogar häufig gemeinsam auftretende Leistungen werden intelligent ergänzt, um einen vollständigen Abrechnungsvorschlag zu generieren [8]. An dieses Niveau kann man sich schrittweise annähern, beginnend mit einfacheren Zuordnungen und dann Regeln/Modelle ausbauen. Wichtig ist auch, **Plausibilitätsprüfungen** einzubauen – z.B. prüfen, ob bei einer

vorgeschlagenen Leistung der entsprechende Zahn überhaupt vorhanden ist, ob Leistungskombinationen zulässig sind etc., ähnlich wie EVIDENT/Doctos bei der Leistungserfassung automatisch prüft [9]. Wenn etwas nicht eindeutig ist, könnte das System den Eintrag als *unsicher* markieren oder eine **Rückfrage** stellen (z.B. "*War die Leistung X am Zahn 14 oder 15?*"), ähnlich den dynamischen Rückfragen in Doctos [10]. Dies wäre allerdings ein fortgeschrittenes Feature – für den Prototyp kann man zunächst davon ausgehen, dass der Arzt im Zweifel selbst korrigiert.

1. **Integration mit der Praxissoftware (Evident)** – Sobald die KI die strukturierten Daten (Befundtext, Leistungsziffern, ggf. Behandlungsempfehlungen) erzeugt hat, müssen diese in *Evident* übernommen werden. Da *Evident* eine lokal installierte Software ist, gibt es hier besondere Herausforderungen. Idealerweise gibt es eine **offene Schnittstelle** oder API. Es lohnt sich, bei Evident anzufragen oder in der Doku nachzusehen, ob z.B. ein Import-Format (XML/CSV) oder eine REST-API existiert. Viele moderne Praxisverwaltungssysteme erlauben den Datenaustausch über Standards wie **HL7** oder bieten Plugins an [11]. Falls Evident eine solche Schnittstelle hat, könnte unser Tool darüber automatisch **Leistungseinträge und Dokumentation einpflegen**. Sollte es keine offizielle API geben, müsste man Workarounds überlegen – etwa über **Robotic Process Automation (RPA)**, die die Maus/Tastatur steuert, um die vorgeschlagenen Leistungen ins Evident einzutragen, oder zumindest die Ausgabe so gestalten, dass eine Helferin sie leicht übertragen kann. In jedem Fall sollte die Integration **getrennt vom KI-Backend** umgesetzt werden, z.B. als lokaler Dienst im Praxisnetz, der die KI-Ausgabe entgegennimmt und an Evident weitergibt. So könnte man z.B. das KI-Backend ggf. in der Cloud laufen lassen, die Ergebnisse aber zunächst an einen lokalen Rechner schicken, der im gleichen Netzwerk wie die Evident-Datenbank ist, um dort die Einträge vorzunehmen. Dass eine solche Integration machbar ist, zeigen bestehende Lösungen: Sowohl Doctos als auch andere KI-Coding-Systeme senden die Ergebnisse **direkt an das jeweilige Verwaltungssystem/EHR** über geeignete Schnittstellen [12] [13]. Unser System sollte also möglichst **automatisiert** die Dokumentation in Evident aktualisieren, um den vollen Nutzen zu erzielen (letztlich träumen wir von einer *nahtlosen* Lösung: Arzt spricht → Eintrag erscheint in Evident ohne weiteren manuellen Schritt).

2. **Datenbank und Wissen** – Möglicherweise wird eine kleine **Datenbank** benötigt, um z.B. Stammdaten vorzuhalten: die Gebührenordnung (BEMA/GOZ-Katalog) zum Nachschlagen der Leistungen, evtl. eine Liste der Patienten/Zähne (könnte aus Evident synchronisiert werden, um z.B. Zahnstatus zu kennen für Plausiprüfung), und um aufgezeichnete Dokumentationen zwischenzuspeichern. Die KI-Auswertung könnte z.B. auch in Evident als Freitext-Dokumentation gespeichert werden, falls eine automatische Leistungsübernahme nicht 100% sicher ist – d.h. das Gesprochene wird als Textnotiz abgelegt (sozusagen als nachvollziehbarer **Audit-Trail**), und parallel werden die erkannten Leistungen als solche markiert. Einige professionelle Coding-Systeme legen sogenannte **Audit Trails** an, die genau erklären, warum eine bestimmte Codierung vorgeschlagen wurde [14] [15]. Für den Anfang kann es genügen, den erkannten Text + vorgeschlagene Codes dem Arzt zur Bestätigung anzuzeigen und dann in Evident zu speichern.

3. **Cloud-Backend vs. lokale Installation** – Hier muss abgewogen werden. Eine **Cloud-Lösung** ist insofern attraktiv, als rechenintensive Aufgaben wie Spracherkennung und LLM-Verarbeitung auf leistungsstarken Servern erledigt werden können. Man könnte z.B. einen Cloud-Server mit GPU nutzen oder sogar Dienste wie OpenAI GPT-4 via API einbinden. Allerdings sind **Datenschutz und Sicherheit** kritische Faktoren: Wir arbeiten mit *Patientendaten*, die in Deutschland unter Datenschutz (DSGVO) und ggf. ärztlicher Schweigepflicht stehen. Sprachdaten könnten sensible Informationen enthalten. Wenn Cloud, dann nur mit **hohen Sicherheitsstandards** (z.B. Anbieter, der HIPAA- oder DSGVO-konform arbeitet, Daten verschlüsselt überträgt und nicht speichert). OpenAI bietet z.B. für Unternehmenskunden

spezielle **HIPAA-konforme Umgebungen** an [16]. Alternativ kann man von Anfang an auf eine **On-Premises-Lösung** setzen: d.h. einen Server in der Praxis installieren, der das Backend hostet (oder sogar alles auf dem iPad lokal – was aber für LLM wahrscheinlich zu schwergewichtig ist). Ein Hybrid-Ansatz wäre: Sprach-zu-Text läuft lokal (es gibt Modelle wie Vosk oder Whisper, die offline funktionieren), und für die komplexe NLP-Auswertung nutzt man ggf. einen Cloud-LLM, wobei man die übertragenen Inhalte anonymisiert (z.B. keine echten Namen verwendet) [17]. In jedem Fall müssen **alle Übertragungen verschlüsselt** erfolgen (HTTPS) und Zugriff nur für berechtigte Nutzer (z.B. nur innerhalb des Praxisnetzwerks oder via VPN).

4. **Benutzerverwaltung & Sicherheit** – Da das Tool zunächst intern im eigenen Team genutzt wird, kann man es simpel halten. Später, falls andere Praxen es testen, sollte ein **Login/ Authentifizierungssystem** vorgesehen werden. Jede Aktion (z.B. Aufzeichnung starten) könnte protokolliert werden, um nachvollziehbar zu machen, welcher Nutzer was dokumentiert hat. Dadurch behält man auch im Blick, falls die KI mal Fehler macht – man kann dann den Prozess auditieren. Die **UI** selbst sollte schlicht und ablenkungsfrei sein: große Aufnahme-Buttons, klare Anzeige der erkannten Texte und Codes, evtl. eine Möglichkeit, erkannte Leistungen abzuwählen oder zusätzliche Infos zu ergänzen (z.B. Zahnnummer, falls nicht erkannt). Hier ist *Usability* wichtig, damit der Workflow den Arzt nicht aufhält.

Abb.: Beispiel eines KI-gestützten Kodierwerkzeugs aus dem medizinischen Bereich. Die gesprochene oder geschriebene Notiz (links) wird von der KI analysiert, welche passende medizinische Codes (hier z.B. Prozedur- und Diagnosecodes) vorschlägt. Solche Systeme nutzen KI/LLM, um unstrukturierte Texte auszulesen und relevante Codes zuzuweisen [18]. In unserem Projekt sollen analog zahnmedizinische Befundtexte in GOZ/ BEMA-Abrechnungspositionen umgewandelt werden.

Technologiewahl: Tech-Stack im Detail

Angesichts der obigen Architektur und der vorhandenen **Skills (Python, LLM, Security)** des Entwicklers, bietet sich folgender *Tech-Stack* an:

- **Programmiersprache & Backend-Framework:** Python ist eine ausgezeichnete Wahl für das Backend. Mit Python hat man Zugriff auf zahlreiche Machine-Learning-Bibliotheken (PyTorch, TensorFlow) und NLP-Frameworks (spaCy, HuggingFace) sowie Schnittstellen für Audioverarbeitung. Für die Web-API kann man ein leichtgewichtiges Framework wie **FastAPI** oder **Flask** nutzen. Diese ermöglichen es, REST-Endpoints oder WebSocket-Verbindungen bereitzustellen, über die das Frontend Audio hochlädt und Ergebnisse abruft. (FastAPI wäre z.B. praktisch, da es async unterstützt – hilfreich für gleichzeitige Audio-Streams – und automatisch ein Web-UI für Tests bereitstellt.)

- **Frontend:** Für den Prototypen reicht eine einfache Weboberfläche (HTML/CSS/JavaScript) [4], die z.B. mit Bootstrap oder einem minimalen JS-Framework (React ist fast schon überdimensioniert, aber könnte genutzt werden) erstellt wird. Wichtig ist vor allem die Fähigkeit, Audio vom Mikrofon aufzuzeichnen. Hier kann die Browser API `getUserMedia` genutzt werden, um Audiodaten zu erfassen. Mit JavaScript lässt sich das Audio in kleine Pakete streamen (WebSockets) oder komplett als Blob nach Aufnahmeende versenden. Auf dem iPad kann diese Web-App im Vollbild laufen. Später, wenn Performance oder iPad-Hardware-Funktionen verbessert genutzt werden sollen, kann man überlegen, eine native App zu schreiben (für iOS z.B. Swift/SwiftUI) oder eine Cross-Platform-Lösung wie **Flutter** oder **React Native** zu verwenden, die ebenfalls Zugriff auf das Mikrofon bieten. Für den Anfang ist jedoch eine Web-Lösung schneller umsetzbar und einfacher zu debuggen.

- **Speech-to-Text (ASR):** Eine Möglichkeit ist, das **Whisper-Modell** von OpenAI einzusetzen. Es gibt Python-Bindings (z.B. via `openai` Paket für die Cloud-API oder lokal via `whisper`/ `transformers`). Whisper hat den Vorteil, dass es **mehrsprachig** und für medizinische Begriffe recht robust ist. Allerdings braucht das große Modell eine GPU, sonst ist es langsam. Für Testläufe könnte man die **OpenAI-API** nutzen (Achtung auf Datenschutz – nur Testdaten verwenden, oder OpenAI Enterprise mit Datenschutzvereinbarung). Alternativ bieten sich Dienste wie **AssemblyAI**, **Google Speech-to-Text** oder **Microsoft Azure Cognitive Services** an, die auch direkt deutsche medizinische Sprache verarbeiten können. Python hat z.B. die Bibliothek `SpeechRecognition`, die man mit Google API Keys nutzen kann, aber für den professionellen Einsatz wäre eine robustere Lösung nötig. Ein weiterer Open-Source-Weg: **Coqui STT** (Nachfolger von Mozilla DeepSpeech) oder **Vosk**. Diese kann man lokal trainieren/fine-tunen auf medizinisches Deutsch, falls nötig. Da der Entwickler dazulernen möchte, wäre es ein tolles Lernprojekt, Whisper lokal zu testen und zu verstehen, wie man ggf. **Modelle fine-tuned** oder Custom Words hinzufügt. Für die *Echtzeitfähigkeit* könnte man auch überlegen, das Audio in kurze Segmente (einige Sekunden) zu schneiden und schrittweise zu transkribieren, damit nahezu Live-Untertitel entstehen, die die KI schon während des Sprechens verarbeiten kann.

- **NLP/LLM Backend:** Hier kommt die Python-Integration eines LLM zum Tragen. Um schnell zu starten, kann man die OpenAI-API (z.B. GPT-4 oder GPT-3.5) ansprechen – damit erzielt man vermutlich direkt recht gute Ergebnisse beim Extrahieren von Leistungen aus Text. Doch langfristig könnte man ein eigenes Modell nutzen oder ein open-source LLM (wie LLaMa 2, GPT-J, etc.) feinjustieren. Für speziell deutsche Dental-Terminologie gibt es vermutlich kein fertiges Modell, daher wäre Fine-Tuning einer bestehenden Architektur denkbar. Alternativ kann man einen zweistufigen Ansatz umsetzen: zuerst mit einem **NLP-Pipeline**-Ansatz bestimmte Keywords erkennen (z.B. mit spaCy eine Entitätserkennung "Leistung" trainieren, oder einfach via Dictionary-Suche), und dann ein kleineres Regelwerk anwenden. Allerdings sind LLMs sehr praktisch, weil sie **Kontext verstehen** – z.B. erkennen sie aus einem Satz wie *"Extraktion 38 wegen tiefer Karies, Lokalanästhesie, Naht, Anweisung zur Kontrolle in einer Woche"* mehrere Einzelleistungen (Extraktion, Anästhesie, Naht, Nachkontrolle) und könnten direkt die entsprechenden Ziffern vorschlagen. Ein gut gebauter Prompt an GPT-4 könnte so etwas bereits leisten. Für die Umsetzung bedeutet das: Python-Server schickt den transkribierten Text an den LLM (entweder via API oder lokal), erhält die analysierte Antwort (z.B. in JSON-Format mit Feldern `{"befund": "...", "leistungen": [...], "planung": "..."}`) zurück und verarbeitet diese weiter. Hier kann der Entwickler viel über **Prompt-Design** und evtl. **Modelltraining** lernen. Sicherheitstechnisch muss man aber aufpassen: Falls Cloud-LLM genutzt wird, keine echten Patientendaten ohne Verschlüsselung oder vertragliche Absicherung schicken.

- **Datenhaltung:** Eine relationale **Datenbank** (z.B. PostgreSQL oder SQLite) kann verwendet werden, um Informationen wie Benutzerkonten, Logs der Dokumentationen, oder Mapping-Tabellen (Leistungsbeschreibungen ↔ Gebührenziffern) vorzuhalten. Für den Anfang könnte man auch einfach in Dateien/JSON loggen, aber perspektivisch ist eine DB sinnvoll. EVIDENT selbst hält die Patientendaten, es wäre aber interessant z.B. die generierten Dokumentationen extern zu speichern (für Auswertung oder falls etwas nicht ins Evident übertragen wurde). Insbesondere wenn man später andere Praxen einbindet, braucht man eine zentrale DB (dann aber wegen Datenschutz evtl. je Praxis eine separate Instanz).

- **Integration mit Evident:** Technisch hängt das von Evident ab. Möglichkeiten:

- Falls Evident **Import-Schnittstellen** bietet: z.B. eine definierte **HL7**-Schnittstelle oder REST-API für Leistungsdaten. Einige AI-Coding-Lösungen setzen auf HL7 or API-Integration, um Codes

direkt in das System zu bekommen [11]. Wir sollten recherchieren oder beim Hersteller nachfragen, ob es eine offiziell unterstützte Variante gibt (in den News war von einer *Schnittstelle* für Doctos die Rede [19], evtl. könnte man an diese Schnittstelle andocken, sofern offengelegt).
- Ohne offizielle API: Nutzung der **Datenbank** von Evident (wenn es eine SQL-basierte DB hat und man sich traut). Dies wäre riskant und evtl. nicht erlaubt, da direkte DB-Eingriffe die Integrität gefährden könnten.
- **Halbautomatische Lösung:** Die KI-Anwendung könnte nach der Spracherkennung dem Nutzer einfach die fertigen Texte und Abrechnungspositionen anzeigen, die dieser per Copy&Paste ins Evident übertragen kann. Das ist für einen Prototyp akzeptabel, reduziert aber den Automatisierungsgrad.
- Ein anderer kreativer Ansatz ist **RPA (Robotic Process Automation)**: Das Programm könnte im Hintergrund die Evident-Oberfläche steuern (Maus, Tastatureingaben simulieren), um die Leistungen einzutragen. Es gibt Tools/Bibliotheken dafür (z.B. PyAutoGUI für einfache Dinge). Für die interne Nutzung mag das funktionieren, ist aber fehleranfällig.

Langfristig wäre die sauberste Lösung, mit Evident zusammen eine Anbindung zu implementieren – vielleicht gibt es auch **Partner-Programme** für Entwickler, um Integrationen zu bauen (Evident hat ja Partnerschaften z.B. mit Nelly, Doctos etc., was zeigt, dass prinzipiell Integrationen möglich sind).

- **Security und Compliance:** Python bietet mit Bibliotheken wie `fastapi` oder `ssl` die Möglichkeit, HTTPS zu erzwingen. Wir sollten **TLS-Verschlüsselung** nutzen, gerade wenn ggf. übers Internet (Cloud) kommuniziert wird. Weiterhin sollten sämtliche sensiblen Informationen (Patientenname, Befunde) wenn möglich **anonymisiert** werden, bevor sie in die Cloud gehen – z.B. könnte man in der Spracheingabe den Patienten gar nicht mit vollem Namen ansprechen. In der Praxis aber sprechen Ärzte oft den Namen nicht dauernd aus, sondern z.B. "der Patient" – das hilft. Für interne Tests ist es trotzdem ratsam, eine sichere Umgebung zu haben. Auch **Zugriffsrechte** müssen bedacht werden: Wenn mehrere Personen das System nutzen, soll z.B. nur der Zahnarzt die Abrechnungsfreigabe machen können, etc. Solche Feinheiten sind evtl. erst später relevant. Unbedingt einhalten sollte man zudem die **gesetzlichen Vorgaben** (in DE: DSGVO, ärztliches Berufsrecht). Das bedeutet, ggf. eine Auftragsverarbeitungsvereinbarung mit einem Cloud-Anbieter schließen, Logs nur so lange aufzubewahren wie nötig, Patienten zustimmen lassen können, wenn Gespräche aufgezeichnet werden (ähnlich wie in Krankenhäusern aufgezeichnete Diktate – normalerweise gibt es dazu Einwilligungen). In den USA denkt man an HIPAA – in unserem Kontext analog die Schweigepflicht und DSGVO: Lösungen wie OpenAI haben Enterprise-Angebote, die HIPAA-konform wären [16], und man kann natürlich jederzeit entscheiden, auf eigene Server auszuweichen [17].

Zusammengefasst könnte der **Stack** also so aussehen:

- *Frontend:* Web (HTML/CSS/JS) für Browser/iPad; später optional native iPad-App.
- *Backend:* Python (FastAPI/Flask) mit WebSocket/REST API.
- *ASR (Speech-to-Text):* z.B. OpenAI Whisper (ggf. via API) oder alternativer STT-Service; langfristig evtl. eigenes Modell.
- *NLP/LLM:* z.B. GPT-4 API in Prototypphase; später feingetuntes Modell oder lokale LLM für Extraktion von Leistungen/Befunden.
- *Database:* PostgreSQL/SQLite for logs, mapping tables, etc.
- *Integration:* falls möglich API/HL7 zu Evident; sonst zunächst manuelle Kontrolle oder RPA.
- *Hosting:* Entweder Cloud-Server (mit Docker-Deployment) oder lokaler Praxisserver; in jedem Fall mit SSL, Auth und Logging.
- *DevOps:* Verwendung von Git für Versionskontrolle, Container (Docker) um die Dienste (ASR Model, Backend) zu isolieren, vielleicht CI/CD Pipeline für Updates, wobei in Praxisumgebung Updates vorsichtig ausgerollt werden müssen (Test -> Prod).

Entwicklungsfahrplan (Roadmap)

Um strukturiert vorzugehen und **während der Entwicklung dazuzulernen**, empfiehlt sich ein schrittweiser Ansatz. Hier eine mögliche Roadmap mit Phasen und Meilensteinen:

1. **Recherche & Anforderungsanalyse:** Zunächst sollten die Anforderungen im Detail festgehalten werden. Hierzu gehört auch, **Evident genauer unter die Lupe** zu nehmen – gibt es dokumentierte Schnittstellen? (Kontakt mit dem Hersteller aufnehmen, Foren durchsuchen). Außerdem die **Gebührenordnungen** (BEMA, GOZ) beschaffen in digitaler Form, um später darauf zugreifen zu können. In dieser Phase kann man auch schauen, was genau Doctos und Sonia anbieten, um sich inspirieren zu lassen. (Erkenntnis bspw.: Doctos fragt nach, wenn Behandlungsdetails fehlen [10], oder Sonia nutzt Vorlagen, um Dokumentationstext gleich in korrekter Form zu haben [20]. Solche Features kann man für später vormerken.)

2. **Minimal Viable Product (MVP) definieren:** Festlegen, welche minimalen Funktionen der erste Prototyp haben muss. Vorschlag: **Sprachaufnahme einer einfachen Behandlungssituation -> Transkription -> Ausgabe eines strukturierten Textes + Liste erkannter Abrechnungscodes** auf einer Test-Webseite. Die Integration in Evident könnte in der MVP-Phase noch manuell erfolgen (z.B. man kopiert die Code-Liste). Wichtig ist, dass dieser Kern funktioniert und akzeptable Genauigkeit bietet.

3. **Umgebung einrichten:** Entwicklungsumgebung aufsetzen (Python Virtualenv, benötigte Libraries installieren: fastapi/flask, pyaudio oder sounddevice für Audio, OpenAI-SDK oder Alternativen, database connectors, etc.). Falls Cloud-Services genutzt werden, API-Schlüssel besorgen. Für das Frontend evtl. einen einfachen dev-Server (node.js oder via Python SimpleHTTPServer) einrichten.

4. **Prototyp Frontend bauen:** Eine sehr simple Seite entwerfen mit einem "Aufnahme Start/Stop" Button und einem Textfeld für Resultate. Hier kann man bereits testen, wie man vom Browser Mikrofonzugriff bekommt. Ein eventueller Stolperstein: Safari (iPad-Browser) hat manchmal Einschränkungen bzgl. Audio-API und WebSocket. Gegebenenfalls zunächst am PC-Browser testen und später am iPad anpassen.

5. **Sprachaufnahme & Transkriptions-Pipeline umsetzen:** Im Backend einen Endpoint einrichten, der Audio entgegennimmt. Für den Start kann es einfach ein POST-Upload einer Aufnahme-Datei sein. Implementiere dann die Anbindung an den STT-Dienst (z.B. rufe Whisper API oder Google STT auf) und erhalte den Text zurück. Gib diesen Text im Backend als Ergebnis zurück. Test: Funktioniert die Kette "sprechen -> Text erscheint" zuverlässig? Hier wirst Du vermutlich schon Feinheiten lernen, z.B. Audioformat-Konvertierung (Web-Aufnahme liefert vielleicht WebM/Opus, aber STT erwartet WAV PCM). Diese technischen Hürden zu meistern, verbessert Deine **Audioverarbeitungs- und API-Kenntnisse**.

6. **Grundlegende NLP-Auswertung implementieren:** Sobald der reine Text da ist, baue eine erste einfache Auswertung. Zum Beispiel, definiere einige Schlüsselbegriffe: "Extraktion" -> Code X, "Krone" -> Code Y, etc., um das Prinzip zu testen. So bekommst Du schnell ein Gefühl, wie der Text strukturiert ist. Parallel könntest Du auch bereits ein LLM einbinden – z.B. schick den Text an GPT-3.5 mit einer Prompt, die die Leistungen extrahiert. Sieh Dir die Antworten an und justiere den Prompt. Das ist ein spielerisches Herantasten an die KI-Komponente. Hier lernst Du **Prompt-Engineering** und ggf. schon etwas **Fehlertoleranz** (die KI wird manchmal Unsinn antworten, sodass Du herausfinden musst, wie du die Frage präziser stellst).

7. **Strukturierte Ausgabe formatieren:** Entwickle ein Datenformat für die Ergebnisse. Etwa ein JSON mit Feldern `{"dokumentation": "Freitext Dokumentation", "codes": [{"code": "GOZ123", "beschreibung": "..."}], "planung": "Empfehlung..."}`. Dieses Format sollte sowohl für Dich intern (Logging, Tests) nützlich sein, als auch später zur Integration. Eventuell zeigst Du die Ergebnisse im Frontend hübsch an: z.B. der erkannte Freitext-Befund, darunter eine Liste von vorgeschlagenen Leistungen mit Checkboxen. So könnte der Zahnarzt durchsehen und z.B. einzelne Vorschläge deaktivieren, falls die KI falsch liegt, bevor es endgültig übernommen wird.

8. **Iterative Verbesserung der KI-Komponente:** Nun beginnt die eigentliche **Feinjustierung**. Teste das MVP mit verschiedenen Beispielszenarien:

9. einfache Prophylaxe-Sitzung,
10. Füllungstherapie,
11. komplexere Sachen wie prothetische Versorgung über mehrere Sitzungen,
12. Notfallbehandlung, etc.

Überprüfe, ob die KI die wichtigen Dinge erkennt. Wahrscheinlich wirst Du feststellen, dass reine Schlüsselwortsuche nicht reicht (z.B. "Krone provisorisch eingesetzt" vs "Krone definitiv eingesetzt" – unterschiedliche Leistungen). Hier kannst Du den LLM-Einsatz verstärken. Evtl. sammelst Du ein kleines *Trainings-Datenset*: nimm aufgezeichnete Beispiel-Sätze und annotiere sie manuell mit den korrekten Codes. Damit könntest Du z.B. ein kleines **Fine-Tuning** eines Modells versuchen oder regelbasierte Muster ableiten. Achte darauf, Kontext zu nutzen: Häufig muss die KI mehrere Sätze zusammen betrachten. In diesem Schritt kannst Du Deine **NLP-Fähigkeiten** ausbauen – vielleicht schaust Du Dir an, wie man Named Entity Recognition oder Textklassifikation in spaCy machen könnte, oder experimentierst mit OpenAI Functions (die neuen GPT-API können strukturierte JSON-Ausgaben erzeugen, was nützlich sein könnte).

1. **Integration mit Evident vorbereiten:** Bevor Du live auf die echte Datenbank gehst, simuliere die Integration. Evtl. erstelle eine **Dummy-Datenbank** oder -Tabelle, wo Du die Ergebnisse hinschreibst, wie Du sie in Evident bräuchtest. Falls z.B. Evident CSV-Import zulässt, baue eine Funktion, die eine CSV mit den Leistungen füllt. Teste diesen Import manuell. Oder falls es eine API gibt, versuche mit Testdaten einen Eintrag zu erzeugen. Wenn Dir RPA der gangbare Weg scheint, probiere es an einer Kopie von Evident (oder nach Feierabend auf echten Daten vorsichtig). Wichtig: In dieser Phase sollte man nichts tun, was die laufende Praxissoftware stört – also immer getrennt oder mit Rücksprache. Möglicherweise stellst Du fest, dass es am einfachsten ist, zunächst **nur den Text und die Codes anzuzeigen**, und die Helferin klickt dann in Evident die vorgeschlagenen Ziffern an. Das wäre dann der v1-Workflow.

2. **Tests im Echtbetrieb (intern):** Ist die Basisfunktion stabil, fang an, das System **im Praxisalltag** testweise zu benutzen. Zunächst vielleicht bei eigenen Patienten und mit dem Wissen, dass es ein Test ist – also die Assistentin schreibt eventuell noch konventionell mit, falls was schiefgeht. Vergleiche die Ergebnisse: Wie genau ist die Transkription in der lauten Praxisumgebung? Versteht das Mikrofon den Arzt auch mit Maske oder wenn er vom Patienten wegschaut? Wie gut decken die KI-Vorschläge die tatsächlichen Leistungen ab? Hier ist mit Feedbackschleifen zu rechnen: Du wirst neue Anpassungen vornehmen (z.B. neues Fachwort hinzufügen, Schwellenwerte ändern, UI verbessern, weil der Touchscreen nicht reagiert mit Handschuhen etc.).

3. **Fehleranalyse & Verbesserung:** Logge unbedingt die Fälle, in denen etwas falsch lief. Vielleicht hat die KI einen Code übersehen – dann lerne daraus und füge eine Regel hinzu oder erweitere die Prompt. Oder die Spracherkennung hat "Infiltrationsanästhesie" als "In Filtration anästhesiert sie" erkannt – dann muss evtl. das Akustikmodell oder eine Nachkorrektur ran. Auch die **Geschwindigkeit** ist ein Thema: Es bringt nichts, wenn der Arzt nach jedem Patienten 2 Minuten warten muss, bis die KI fertig ist. Whisper braucht evtl. so lange – also ggf. nach Optimierungen suchen (kleineres Modell, oder Zwischenergebnisse schon anzeigen).

4. **Erweiterung: Dynamische Vorlagen und Planung:** Sobald die Kernfunktion (Befund + Abrechnung) funktioniert, kann man die **Planungs-Komponente** ausbauen. Das könnte bedeuten, dass die KI auch *Empfehlungen für die weitere Behandlung* notiert (z.B. "in 6 Monaten zur Kontrolle", "Überweisung KFO erwägen" etc.). Diese Informationen ließen sich evtl. ebenfalls in Evident eintragen (z.B. als Wiedervorlage oder Bemerkung). Auch könnte man an **Behandlungsvorlagen** denken: In Evident gibt es wahrscheinlich Textbausteine oder Vorlagen. Sonia zum Beispiel nutzt über 100 optimierte Vorlagen für Dokumentationen und kann diese automatisch befüllen [20]. Man könnte die KI so trainieren, dass sie gewisse Standardsätze verwendet (gleichmäßige Dokumentation unabhängig vom Sprecher [21]). Dies erhöht die Qualität und Einheitlichkeit. Für Dich als Entwickler hieße das, evtl. ein **Template-System** zu integrieren: z.B. je nach erkannter Behandlung bestimmte Textmodule in die Dokumentation einbauen lassen, damit es juristisch korrekt formuliert ist. Dieser Schritt ist optional, aber für den Praxisnutzen groß: Einheitliche, fehlerfreie Dokumentationen ohne Extra-Aufwand [22].

5. **Benutzerfreundlichkeit und UI-Feinschliff:** Jetzt, da es läuft, verfeinere die Bedienung. Vielleicht baust Du eine **Sprachsteuerung** ein, sodass der Arzt via Sprachbefehl die Aufnahme starten/beenden kann (um ganz freihändig zu sein). Oder Du stellst sicher, dass die Schrift groß genug ist, Dunkelmodus etc. – Dinge, die im Alltag wichtig werden. Eventuell implementiere eine **Fehlerkorrektur**-Möglichkeit: Wenn die KI etwas falsch verstanden hat, sollte der Nutzer es leicht korrigieren können (z.B. durch Antippen und Editieren des Textes, oder Auswahllisten für Codes).

6. **Sicherheitsüberprüfung & Datenschutz:** Vor einem breiteren Einsatz unbedingt checken: Werden irgendwo Patientendaten ungeschützt gespeichert? Sind Logs anonym? Läuft das System nur im Praxisnetz oder ist es übers Internet erreichbar? Gegebenenfalls härte das System: z.B. Passwortschutz für die Web-App, regelmäßige Software-Updates (gerade wenn bekannte Libraries mit Sicherheitslücken genutzt wurden, updaten). Falls Cloud-Services im Spiel sind, **Verschlüsselung** sicherstellen. In dieser Phase könnte man auch einen **Penetrationstest** in Betracht ziehen (zumindest intern versuchen, die eigene API zu hacken), um Schlupflöcher zu finden. Da "Security" eine deiner Kern-Skills ist, kannst Du dein Wissen hier direkt anwenden, indem Du z.B. OWASP-Richtlinien für Web-Anwendungen gegenprüfst.

7. **Pilotphase mit anderen Nutzern:** Wenn die interne Testphase erfolgreich war und das Team sich an das Tool gewöhnt hat (idealerweise: die Dokumentation läuft "nebenbei" per Sprache und keiner mag mehr ohne arbeiten), kann man darüber nachdenken, das System auch *anderen Praxen* zum Test anzubieten. Vorher wäre zu klären, wie man es verteilt (Cloud-Service mit getrennten Accounts vs. Installation vor Ort in jeder Praxis). Für eine Pilotphase könntest Du befreundete Kollegen fragen. Davor unbedingt Daten zurücksetzen oder Dummy-Daten verwenden – Externe sollten keine echten internen Protokolle sehen. Das Feedback externer User wird weitere Verbesserungspunkte liefern (jede Praxis hat leicht andere Abläufe, evtl. andere Software außer Evident, etc.). Sollte das System allgemein funktionieren, wäre das auch der Punkt, über *Produktisierung* nachzudenken (z.B. Branding, Webseite, Support-Strukturen).

8. **Langfristige Vision:** Mit den gewonnenen Erkenntnissen kannst Du die Roadmap erweitern. Denkbar sind Features wie **Echtzeit-Entscheidungsunterstützung** (während der Arzt spricht, schlägt die KI z.B. vor: "Soll gleichzeitig der Zahnstein entfernt werden?"), oder **Analytics** (das System wertet aus, welche Leistungen oft nicht erwähnt werden und gibt Tipps). Auch **mehrsprachige Unterstützung** (für fremdsprachige Patienten oder Ärzte) oder **Diktate für Briefe** etc. wären Erweiterungen. Das geht jedoch über das Kernprojekt hinaus. Zunächst ist wichtig, den Fokus zu halten: eine zuverlässige, sichere Basislösung für die Sprachdokumentation und Abrechnung zu schaffen.

Lernaspekte und Skill-Entwicklung

Während dieser Projektentwicklung wirst Du in mehrfacher Hinsicht dazulernen und Deine bestehenden Kenntnisse ausbauen können:

- **Python-Expertise:** Durch die Implementierung des Backends (API-Server, Datenbankanbindung, etc.) vertiefst Du deine Erfahrung in Python-Webentwicklung. Umgang mit Async-Programmierung (für Audio-Streaming), Performance-Tuning, und Nutzung von ML-Bibliotheken gehören dazu. Auch das Packaging/Deployment (z.B. als Docker-Container) wird Deine DevOps-Kenntnisse erweitern.

- **Umgang mit LLMs/NLP:** Du wirst praktisch anwenden, was es heißt, einen LLM für einen konkreten Anwendungsfall einzusetzen. Von der Auswahl eines Modells über Prompt-Engineering bis hin zum möglichen Fine-Tuning auf Domain-Daten – all das sind wertvolle Erfahrungen. Gerade medizinische NLP ist ein speziales Feld; durch das Experimentieren (z.B. mit spaCy Pipelines oder HuggingFace Transformers) baust Du spezialisiertes Know-how auf. Außerdem lernst Du, KI-Ausgaben kritisch zu bewerten und mit Unsicherheit umzugehen (Stichwort human-in-the-loop, wenn der Arzt doch nochmal drüberschaut).

- **Sicherheitspraktiken:** Da Du von Anfang an die **Security-Aspekte** berücksichtigen musst, wendest Du Dein Wissen in der Praxis an. Du planst ein System, das Patientendaten verarbeitet – also lernst Du ggf. mehr über **Verschlüsselung, Zugriffskontrolle, sichere Konfiguration** von Webservern. Wenn Du z.B. einen Pen-Test machst oder Bedrohungsanalysen (Threat Modeling) für dein System, stärkt das Deine Fähigkeiten, sichere Software zu entwickeln. Auch das Thema **Datenschutz** (legal und organisatorisch) kommt hier ins Spiel – eine Facette der Security, die oft genauso wichtig ist.

- **Projekt- und Produktmanagement:** Indem Du die Roadmap erstellst und abarbeitest, schulst Du Dich auch in **Planung und agilem Vorgehen**. Du lernst Prioritäten zu setzen (welches Feature zuerst, was ist MVP) und mit Nutzerfeedback umzugehen. Wenn Du extern testest, sammelst Du Erfahrung im **Deployment** eines Produkts in einer echten Umgebung und im **Support** (die Praxen werden Fragen/Probleme haben, die Du lösen musst).

- **Domain-Wissen Zahnmedizin/Abrechnung:** Unvermeidlich wirst Du tiefer in die Materie der zahnärztlichen Abrechnung eintauchen. Beim Verknüpfen von Befund und Leistung lernst Du die Gebührenziffern, ihre Bedingungen und häufigen Kombinationen. Dieses Wissen hilft Dir wiederum, Dein KI-System zu verbessern (denn je besser Du die Logik verstehst, desto gezielter kannst Du die KI trainieren). Am Ende bist Du nicht nur tech-seitig, sondern auch *fachlich* deutlich versierter – ein großer Vorteil, wenn Du das Tool weiterentwickelst oder vielleicht auch anderen Ärzten erklärst.

Fazit

Zusammengefasst besteht die Planung darin, ein **mehrschichtiges System** aufzubauen, das von der **Sprachaufnahme bis zur fertigen Abrechnungsdokumentation** alles abdeckt. Technologisch setzt Du auf einen **Python-basierten KI-Backend** mit Speech-to-Text und LLM-Komponenten, einer leichtgewichtigen **Frontend-Lösung** fürs iPad, und einer wohlüberlegten **Integration** in die bestehende Praxissoftware. Während der Entwicklung wird in kleinen Schritten vorgegangen – beginnend bei einfachen Prototypen bis hin zur robusten Lösung –, was es ermöglicht, stetig dazuzulernen und Risiken zu minimieren.

Die Idee, zunächst intern zu testen und erst bei Stabilität weitere Nutzer einzubeziehen, ist genau richtig: So stellst Du sicher, dass das Produkt wirklich **Mehrwert** bringt, bevor es verbreitet wird. Wenn alles gelingt, wirst Du am Ende ein Tool in den Händen halten, das dem Zahnarzt erlaubt, **ohne Assistenz während der Behandlung per Sprache Befunde, Dokumentation und Leistungen zu erfassen**, und das **automatisch vollständige Abrechnungsvorschläge** erzeugt [1] [8]. Die erwarteten Vorteile – enorme Zeitersparnis, vollständige Dokumentation, bessere Abrechnung – sind den Aufwand wert. Und nicht zuletzt wird dieses Projekt Dir persönlich helfen, Dich als Entwickler erheblich weiterzuentwickeln, indem es cutting-edge Technologien (Spracherkennung, KI, medizinische Software) in einem sinnvollen realen Anwendungsfall vereint. Viel Erfolg bei der Umsetzung!

Quellen: Praktische Erfahrungen und Herstellerinformationen zu KI-gestützter medizinischer Dokumentation [1] [8], Beispiel-Architekturen für AI Medical Scribe Systeme [4] [16], sowie Marktübersichten zu automatisierter medizinischer Kodierung [11] [7] und Erfolgsberichte existierender Dental-Sprachassistenz-Lösungen [23] [20] (zur Veranschaulichung der Machbarkeit und Nutzen).

---

[1] [8] [9] [10] Exklusiv: EVIDENT bindet Doctos an - EVIDENT GmbH
https://www.evident.de/software/exklusiv-evident-bindet-doctos-an/

[2] [3] [20] [21] [22] [23] Sonia
https://sonia.so/

[4] [5] [6] [13] [16] [17] Transforming Clinical Documentation: Building an AI Medical Scribe with LLMs | by Lotus Labs | Jun, 2025 | Medium
https://lotuslabs.medium.com/transforming-clinical-documentation-building-an-ai-medical-scribe-with-llms-a385bd0cdb17

[7] [11] [12] [14] [15] [18] AI Medical Coding Software | Belitsoft
https://belitsoft.com/ai-medical-coding-software

[19] EVIDENT bindet Doctos an - zm-online
https://www.zm-online.de/markt/marktanzeigen/detail/evident-bindet-doctos-an