

Seminarfacharbeit Klasse 10
Schuljahr 2024/25

Schneller, besser, effizienter Entwicklung eines File Explorers mit schneller Suchfunktion

Fachbetreuer: Herr Süpke
Informatikbetreuer: Herr Süpke
Seminarfachbetreuer: Frau Dr. Moor

Schüler: Nino Fischer (10spa)
Jessica Nolle (10spa)
Magnus Schultheis (10spb)

Datum: 28. Mai 2025, Erfurt

Inhaltsverzeichnis

1	Einleitung	2
2	Grundlagen von Dateisystemen	3
3	Datenbanken als Speichermedien	3
4	Methoden zur Suchergebnisauswahl	4
4.1	Bestimmung der Ähnlichkeiten von Zeichenketten mithilfe der Levenshtein-Distanz	4
4.2	Worteinbettungen mithilfe neuronaler Netze	5
4.2.1	Funktionsweise von Worteinbettungen	5
4.2.2	Eigene Implementation des Skip-Gram-Modells	6
5	Genutzte Softwareressourcen	8
5.1	Entwicklung einer Benutzeroberfläche mit Tauri	8
5.2	Auswahl einer geeigneten Datenbanktechnologie	9
5.3	Evaluierung von Parallelisierungstechnologien	9
6	Umsetzung des Dateiverwalters	10
6.1	Strukturierung des Programms	10
6.2	Aufbau der Benutzeroberfläche und interne Kommunikation	11
6.3	Visualisierung des Dateisystems	13
6.4	Realisierung der Datenbankgenerierung	13
6.5	Aktualisieren der Datenbank bei Dateisystemänderungen	14
6.6	Verwirklichen der Dateisuche	15
6.7	Essentielle Funktionen für den Nutzer	16
7	Vergleich des Semi-Explorers mit dem Windows-Explorer	17
8	Zusammenfassung	19
9	Literatur- und Quellenverzeichnis	20
	Literaturverzeichnis	20
	Quellenverzeichnis	20
	Abbildungsverzeichnis	21

1 Einleitung

Das Sprichwort „Zeit ist Geld“ stammt von Benjamin Franklin und ist 1748 [29] in seinem Buch „Ratschläge für junge Kaufleute“ erstmals erschienen. Es bedeutet, dass Zeit so wertvoll wie Geld ist, sie effizient eingesetzt und nicht verschwendet werden sollte.

Die Suche nach einer Datei (engl. file) kann in elektronischen Medien, wie File Explorern, viel Zeit beanspruchen. Beispielsweise sind die Suchergebnisse im Windows-Explorer zum Teil nicht treffend und zeigen Dateien bei möglichen Rechtschreibfehlern nicht mehr an. Eine Umfrage des Startups amberSearch aus dem Jahr 2022 [2] hat ergeben, dass Mitarbeiter täglich bis zu 30 Minuten [2] zur Suche unternehmensinterner Informationen benötigen. Dies ergibt bei circa 230 Arbeitstagen pro Jahr [2] fast eine ganze Arbeitswoche. Entsprechende Studien aus dem Schulalltag sind nicht bekannt. Wir können jedoch aufgrund der eigenen mehrjährigen Erfahrungen mit dem Windows-Explorer eine zeitintensive Dateisuche bestätigen. Selbst wenn man beim Schüler nur eine wöchentliche Suchzeit von 30 Minuten annimmt, ergibt dies einen ganzen Tag Suchzeit pro Jahr. Es ist davon auszugehen, dass die digitale Suchzeit in den nächsten Jahren bei dem aktuell exponentiellen Wachstum der Datenmengen weiter steigen wird.

Wir wollten dieses im alltäglichen Umgang mit dem Windows-Explorer auftretende Problem und die damit einhergehende Zeitverschwendung nicht weiter hinnehmen und waren motiviert, uns diesem Thema zu widmen. Aus diesem Grund setzen wir uns das Ziel, mit dieser Seminarfacharbeit einen eigenen File Explorer (nachfolgend Semi-Explorer benannt) zu erstellen, der sich durch eine schnellere Suchfunktion auszeichnet.

Der erste Teil unserer Arbeit widmet sich der Theorie zu File Explorern und ihren Funktionen, um ein Verständnis für deren Verwendung zu vermitteln. Des Weiteren wird dargelegt, wie eine effiziente Suche anhand von Datenbanken, der Levenshtein-Distanz und unserem neuronalen Netz innerhalb des Semi-Explorers erreicht werden soll. Zusätzlich soll dargelegt werden, welche Programmiersprachen und Systemressourcen zur Entwicklung unseres Semi-Explorers genutzt werden. Im zweiten Teil veranschaulichen wir unseren Eigenanteil und gehen detailliert auf unser Programm, den Semi-Explorer, ein. Es soll deutlich gemacht werden, wie wir die verbesserte Suche des Semi-Explorers mit unserem Programm umsetzen und garantieren können.

Zur Erreichung unserer Ziele haben wir uns zunächst eingehend mit der Fachliteratur zu File Explorern beschäftigt. Des Weiteren konnten wir ermitteln, welche Programmiersprachen sich für unser Vorhaben am besten eignen. Die Entscheidung fiel unter anderem auf die Programmiersprache Rust, welche sich durch Schnelligkeit, Zuverlässigkeit und Speichereffizienz auszeichnet. Im Anschluss daran erfolgte die Aneignung dieser Sprache. Die zuvor theoretisch gewonnenen Erkenntnisse sollten genutzt werden, um einen eigenen File Explorer mit schneller Suchfunktion zu entwickeln.

Wir möchten an dieser Stelle unserem Fach- und Informatikbetreuer Herrn Süpke unseren besonderen Dank aussprechen, der uns stets bei Fragen unterstützte. Außerdem danken wir unserer Seminarfachlehrerin Frau Dr. Moor, insbesondere für die Hinweise im Rahmen der Rhetorikkurse.

2 Grundlagen von Dateisystemen

Eine Datei ist in der Informatik ein Datensatz in Text, Bild, Ton, Audio, Video oder anderer digitaler Form, der auf einem Datenträger gespeichert und durch einen Namen identifiziert wird. Ein File Explorer oder auch Datei-Explorer genannt, ist ein Programm zur Verwaltung von Dateien. Der Begriff „Explorer“ stammt aus dem Lateinischen und bedeutet so viel wie „Entdecker“. Das Dateisystem definiert die Organisation, auf welche Art und Weise Daten auf dem Computer abgespeichert werden können. Der File Explorer listet alle verfügbaren Ordner auf und ermöglicht es, diese zu öffnen und die darin enthaltenen Dateien und Inhalte einzusehen. Zusätzlich kann man mit einem File Explorer auf alle auf dem Computer gespeicherten Dateien in einer Verzeichnisstruktur zugreifen und diese verwalten. Diese Dateien besitzen sowohl einen Dateinamen als auch Attribute, durch welche man entsprechend nähere Informationen über sie erfahren kann. Ein Dateisystem ist folglich dazu fähig, einen Namensraum zu bilden. Dadurch ist das System in der Lage, Dateien zu identifizieren, zu lesen und zu verändern. Eine weitere Funktion des File Explorers ist, dass das Navigieren durch die Festplatte wesentlich einfacher sowie übersichtlicher gestaltet wird.

3 Datenbanken als Speichermedien

Ein wichtiges Fundament für den Semi-Explorer stellt die digitale Datenbank dar. Eine Datenbank ist ein digitales System zur strukturierten Speicherung und effizienten Organisation großer Datenmengen, welche für einen File Explorer essenziell sind. Datenbanken können im Allgemeinen in verschiedene Typen unterteilt werden. Diese unterscheiden sich in der Datenspeicherung und -verwaltung. In dieser Arbeit sind jedoch nur relationale Datenbanken, wie SQLite, von Relevanz. Informationen werden hierbei in tabellarischer Form gespeichert, was einen schnellen Zugriff und eine effiziente Verwaltung ermöglicht.

Mithilfe von Datenbanken kann man unter anderem durch spezielle Suchalgorithmen Informationen schnell suchen und abrufen. Des Weiteren ermöglichen Datenbanken eine einfache und schnelle Aktualisierung von Informationen. Gleichzeitig stellen sie die Datenintegrität sicher, also die Korrektheit, Vollständigkeit und Konsistenz der Daten. Zusätzlich weisen Datenbanken eine große Sicherheit auf, da sie nur berechtigten Personen Zugriff auf Informationen gewähren. Durch Sicherheitsmaßnahmen, wie beispielsweise Zugriffskontrollen und Verschlüsselungen, können sensible Daten geschützt werden.

Damit diese Funktionen erfüllt werden können, bedarf es einer Datenbank, die über bestimmte Datenstrukturen, Schlüssel und Abfragesprachen verfügt. Unter Datenstrukturen werden alle Tabellen, Dokumente oder Graphen zusammengefasst, in welchen Informationen gespeichert werden. Schließlich werden bei den benötigten Schlüsseln zwischen Primärschlüsseln, die zur eindeutigen Identifizierung von Datensätzen gedacht sind, und Fremdschlüsseln unterschieden. Fremdschlüssel werden zur Herstellung von Beziehungen zwischen verschiedenen Tabellen oder Datensätzen benutzt. Abfragesprachen sind Programmiersprachen, welche speziell für den Umgang mit Datenbanken entwickelt wurden. Mithilfe von Abfragesprachen werden Informationen aus Datenbanken effizient abgerufen und verwaltet. Die Standardabfragesprache, auch bekannt als Structured Query Language (SQL), ermöglicht dabei das Aktualisieren, Schreiben und Löschen von Daten.

4 Methoden zur Suchergebnisauswahl

4.1 Bestimmung der Ähnlichkeiten von Zeichenketten mithilfe der Levenshtein-Distanz

Die Levenshtein-Distanz ist eine Abstandsfunktion, eine String-Metrik, welche zur Messung der Distanz zwischen zwei Zeichenketten dient. Die Levenshtein-Distanz oder auch Levenshtein-Ähnlichkeit genannt, beschreibt die minimale Anzahl an Änderungen zwischen zwei Wörtern, die nötig sind, um ein Wort in ein anderes zu ändern. Im Rahmen dieses Prozesses wird aus der ersten Zeichenkette eine zweite generiert. Das Hinzufügen, Entfernen oder Austauschen von Zeichen wird als eine Änderung der Zeichenkette betrachtet. Die Levenshtein-Distanz wurde nach dem Mathematiker Vladimir Levenshtein benannt, der sie 1965 [18] definierte. Der Algorithmus zur Berechnung der Levenshtein-Distanz wird als Levenshtein-Algorithmus bezeichnet.

Die Berechnung der Levenshtein-Distanz unterliegt spezifischen Regeln. Die maximale Länge der Levenshtein-Distanz ist immer die des längeren Strings. Die Levenshtein-Distanz zwischen einem fünfbuchstabigen und einem siebenbuchstabigen Wort kann beispielsweise nicht größer als sieben sein.

Identische Begriffe weisen eine Levenshtein-Distanz von 0 auf, da keine Änderungen nötig sind, um den ersten String in einen anderen zu transformieren. Negative Werte einer Levenshtein-Distanz sind nicht möglich. Die Levenshtein-Distanz gibt alle erforderlichen Operationen zur Änderung von Zeichenketten an. Es werden entweder Änderungen bei unterschiedlichen Wörtern oder keine Änderungen bei gleichen Wörtern vorgenommen. Dementsprechend können diese Änderungen niemals negativ sein.

Im Rahmen der Suche nach einer Datei im Semi-Explorer erfolgt ein Abgleich des Suchbegriffs mit den Datenbankeinträgen. Die Unterschiede zwischen dem jeweiligen Suchbegriff und den Zeichenketten werden dabei mit der Levenshtein-Distanz gemessen. Die Levenshtein-Distanz ist insbesondere dann hilfreich, wenn sich der Benutzer bei der Eingabe des Suchbegriffs verschrieben hat oder nicht mehr genau weiß, wie er die Datei benannt hat. Der Suchalgorithmus kann aufgrund der Levenshtein-Distanz auch solche Dateien finden und trägt zur Steigerung der Benutzerfreundlichkeit bei. Das folgende Beispiel veranschaulicht die Suche nach der Datei "Levenshtein". Es werden jedoch absichtlich zwei Tippfehler eingebaut und statt des Wortes "Levenshtein" wird das Wort "Levinstein" in das Suchfeld eingegeben. Trotz der fehlerhaften Schreibweise identifiziert der Algorithmus die gesuchte Datei. Die Levenshtein-Distanz zwischen den Begriffen "Levinstein" und "Levenshtein" beträgt in diesem Fall 2, wie nachfolgend auch aufgezeigt ist:

0. Levinstein (keine Änderung, Ausgangswort),
1. Levenstein (Ersetzung des Buchstaben i durch den Buchstaben e),
2. Levenshtein (Einfügung des Buchstaben h).

4.2 Wortembeddings mithilfe neuronaler Netze

4.2.1 Funktionsweise von Wortembeddings

Das Prinzip der Wortembedding beschreibt die Idee, Wörter oder Zeichen in Zahlen darzustellen, damit der Computer besser mit ihnen arbeiten kann. Zu diesem Zweck werden verschiedene Techniken genutzt, darunter die One-Hot-Kodierung, bei welcher jedem Wort (Token) eine andere Zahl zugewiesen wird. Der Rechner ist zwar hierdurch in der Lage, die Unterschiedlichkeit der beiden Wörter zu identifizieren und sie anhand ihrer zugewiesenen Nummer auseinanderzuhalten. Es ist ihm jedoch nicht möglich, Aussagen über die kodierten Wörter zu treffen.

In vereinfachter Form sind neuronale Netze Funktionen mit Multiplikatoren für jeden Parameter, die Gewichte genannt werden. Das Trainieren von Modellen ist das Optimieren dieser Gewichte zum Erreichen des angestrebten Ergebnisses. Die Nutzung eines neuronalen Netzes stellt eine mögliche Methode dar, um den Zusammenhang zwischen verschiedenen Wörtern zu veranschaulichen. Die zugrundeliegende Idee besteht darin, dass ähnliche Wörter oder Wörter, die oft zusammen beziehungsweise im ähnlichen Kontext vorkommen, eine kleinere numerische Distanz zueinander haben als Wörter, die in völlig anderen Kontexten verwendet werden.

Im Rahmen der Vektoreinbettung, die in verschiedenen Modellen des Trainierens von Algorithmen mit Problemlösung ohne feste Regeln Anwendung findet (Maschinelles Lernen), wird ein Token zu einer Sammlung von Zahlen (Vektoren) umgewandelt. Die Wörter, welche eine ähnliche Bedeutung haben oder in einem ähnlichen Kontext genutzt werden, liegen mathematisch dicht zusammen. Hierbei werden die Wörter in einen Vektor umgewandelt, um mehrere sich unterscheidende Bedeutungen des gleichen Wortes in unterschiedlichen Kontexten darzustellen. Die Anzahl dieser Kontexte oder Zahlen pro Vektor wird als Dimension des Modells bezeichnet. Sie hat einen großen Einfluss auf die Dauer des Trainings sowie auf die Qualität der Ergebnisse des Modells. Eine weitere wichtige Eigenschaft der Modelle zur Vektoreinbettung ist die Fenstergröße, welche die Anzahl an Wörtern angibt, die das Modell während des Trainings als Kontext nutzt.

Zwei sehr bekannte Vertreter dieser neuronalen Netze sind das Skip-Gram-Modell und das Continuous Bag-of-Words Modell (CBOW). Diese beiden Modelle arbeiten umgekehrt zueinander. Das Skip-Gram-Modell wird trainiert, indem ein Token genutzt wird, um die umliegenden Wörter vorherzusagen. Diese Vorhersage erfolgt gemäß Abbildung 1. Das CBOW-Modell hingegen wird trainiert, indem es basierend auf den umliegenden Wörtern ein Wort vorhersagt, wie in Abbildung 2 dargestellt.

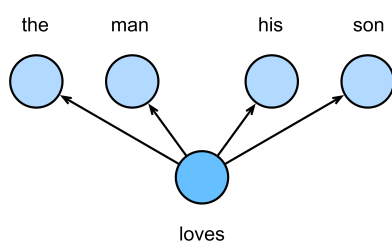


Abbildung 1: Funktionsweise des Skip-Gram-Modells [44]

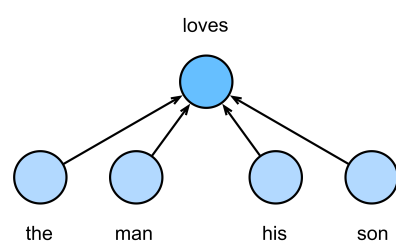


Abbildung 2: Funktionsweise des CBOW-Modells [43]

Zunächst wurden das CBOW-Modell und das Skip-Gram-Modell in Betracht gezogen, das CBOW-Modell wurde aber verworfen. Obwohl dieses Modell weniger Trainingsdauer benötigt hätte, wäre dafür aber auch ein größerer Datensatz vonnöten gewesen. Darüber hinaus weist es eine höhere Kompetenz im Verstehen von Syntax, also der Zusammensetzung von Wörtern, auf. Die Kompetenz ist jedoch von geringerer Relevanz als die Vorteile des Skip-Gram-Modells. Das Skip-Gram-Modell zeigt nämlich eine höhere Präzision bei der Bestimmung von semantischen Zusammenhängen auf, also beim Erkennen von ähnlichen Bedeutungen. Die höhere Präzision der Semantik ist für den Semi-Explorer von signifikanter Bedeutung, da die Implementierung komplementär zur Levenshtein-Ähnlichkeit geschehen soll. Die Levenshtein-Ähnlichkeit liefert bereits Ergebnisse, die grammatikalisch ähnlich sind. Außerdem wurde die Annahme getätigt, dass aufgrund der großen Popularität des Computers und den damit einhergehenden unterschiedlichen Nutzern sowie deren unterschiedlichen Namensgebungen die höhere Kompetenz von seltenen Wörtern von größerer Relevanz ist.

4.2.2 Eigene Implementation des Skip-Gram-Modells

Der erste Schritt des Trainierens ist das Tokenisieren der Trainingsdaten, also das Aufteilen des Datensatzes in sinnvolle Abschnitte. In diesem Fall erfolgt dies in Form von Wörtern. Reguläre Ausdrücke nutzen syntaktische Regeln, um eine Menge an Zeichenketten zu beschreiben. Die genaue Implementierung ist mithilfe eines solchen Regulären Ausdrucks umgesetzt worden, welcher alle aufeinanderfolgenden Buchstaben zusammenfasst und zurückgibt. Nach dem Tokenisieren der Trainingsdaten ist der nächste Schritt das Aufbauen des Vokabulars. Dieses beinhaltet die am häufigsten auftretenden Wörter im Datensatz, welche in einer JSON-Datei mit einem zugehörigen Index gespeichert werden. JSON steht für JavaScript Object Notation und ist ein kompaktes Datenformat in lesbarer Textform.

Die für die Seminarfacharbeit trainierten Modelle haben eine Fenstergröße von fünf und ein Vokabular von 50 000 Tokens und wurden alle mit 20 Epochen trainiert. Dies bedeutet das 20-mal über die Trainingsdaten iteriert wurde. Die trainierten Modelle wurden auf den Wikipedia-Daten der Universität Leipzig mit 1 000 000 [24] Sätzen trainiert. Insgesamt gibt es sechs Modelle. Drei der Modelle wurden mit deutschen und drei mit englischen Daten trainiert. Sowohl die deutschen als auch die englischen Daten beinhalten jeweils 75, 150 und 300 Dimensionen. Im folgenden Teil der Arbeit wird lediglich auf das Semi-Modell Bezug genommen, welches das englische Modell mit 300 Dimensionen darstellt. Das Semi-Modell verwendet die Verlustfunktion Kreuzentropie, die ein Maß für die Qualität eines Modells ist, mit dem Optimierungsalgorithmus Adam, um die Gewichte zu optimieren. Darüber hinaus wurde negatives Sampling verwendet, bei welchem zunächst schlechte Kontextpaare gebildet werden. Anschließend wird versucht, diese Paare mithilfe von Log-Likelihood, einem Algorithmus, zu minimieren. Log-Likelihood gibt an, wie gut ein Modell die beobachteten Daten erklärt. Dies erfolgt, indem die Wahrscheinlichkeit berechnet wird, die gleichen Daten mit anderen Parametern zu erhalten.

Das Modell wurde 170 Minuten lang auf einem Computer mit einer Grafikkarte, der AMD Radeon RX 6600 trainiert. Das Semi-Modell startete mit einem durchschnittlichem Verlust von 9,34 und erreichte einen Verlust von 2,94. Der Verlust ist das Ergebnis der Verlustfunktion. Je geringer der Verlust ist, umso bessere Ergebnisse liefert das Modell.

Eine Darstellung der Vektoreinbettungen ist in Abbildung 3 dargestellt, bei der die 300 Dimensionen des Semi-Modells auf zwei reduziert werden. Die x- und y-Achsen spiegeln hierbei die Ähnlichkeiten untereinander wider. Zur Erstellung des Diagramms wurden zunächst vier Wörter vorgegeben und dann die nächstgelegenen Wörter ermittelt. Anschließend wurden die Wörter dem Diagramm hinzugefügt. Hierbei ist im Diagramm ersichtlich, dass Wörter in unmittelbarer Nähe einen inhaltlichen Zusammenhang haben.

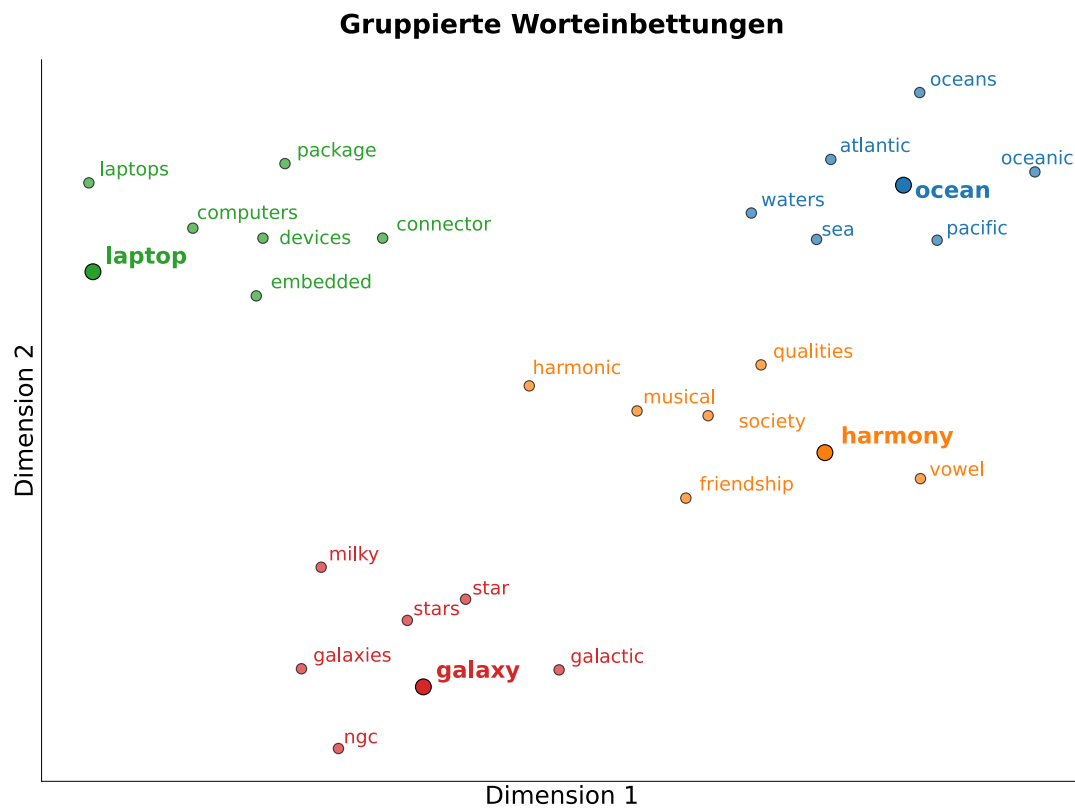


Abbildung 3: Vereinfachte Darstellung von Worteinbettungen von naheliegenden Wörtern in zwei Dimensionen[42]

Für die Anwendung des Modells kann die Kosinus-Ähnlichkeit verwendet werden, welche die Ähnlichkeit zweier Vektoren berechnet. Die Kosinus-Ähnlichkeit zweier Vektoren ist der Kosinus des eingeschlossene Winkel zwischen den Vektoren oder mathematisch das Skalarprodukt. Sie liegt zwischen -1 und 1, wobei eine Kosinus-Ähnlichkeit von 1 bedeutet, dass die Vektoren gleich sind. -1 bedeutet, dass die Vektoren entgegengesetzt sind und 0, dass sie unabhängig voneinander sind.

Vorgegebenes Wort	Wort 1	Ähnlichkeit	Wort 2	Ähnlichkeit
laptop	usb	0.3983	computers	0.3736
harmony	vowel	0.3759	musical	0.3653
galaxy	galaxies	0.5578	milky	0.5288
ocean	atlantic	0.5690	pacific	0.5475

Tabelle 1: Kosinus-Ähnlichkeit der zwei ähnlichsten Wörter aus Abbildung 3

5 Genutzte Softwareressourcen

5.1 Entwicklung einer Benutzeroberfläche mit Tauri

Bei Tauri handelt es sich um ein Framework. Ein Framework gibt die Struktur des künftigen Softwareprodukts vor. Es beinhaltet den Aufbau, wesentliche Funktionen und weitere Elemente. Tauri ist dazu in der Lage, kleine und schnelle Binärdateien für alle wichtigen Desktop- und Mobilbetriebssysteme zu erstellen. Damit kann der Einsatz des Semi-Explorers auf einer Vielzahl von Computern gewährleistet werden, ohne dass eine neue Implementierung nötig ist.

Für das Frontend, also die Präsentationsebene des Programms mit welcher der Nutzer interagieren kann, werden die Skriptsprachen Hypertext Markup Language (HTML), Javascript und Cascading Style Sheets (CSS) verwendet. Die Funktion von HTML besteht grundlegend in dem Aufbau von Webseiten. Das Design wird hingegen mittels CSS definiert, während Javascript die interaktive Veränderung von Inhalt und Design ermöglicht.

Das Backend ist das technische System, welches im Hintergrund abläuft. Dieses kommuniziert mit dem Frontend und sendet beispielsweise im Semi-Explorer Suchergebnisse zum Frontend, welche zuvor ermittelt worden. Der Nutzer kann sie nun betrachten. Genauso kann das Frontend aber auch mit dem Backend kommunizieren. Im Semi-Explorer kann der Nutzer zum Beispiel durch einen Knopfdruck diese Suchergebnisse anfordern. Für die Implementierung des Backends kann zwischen den drei Programmiersprachen Rust, Kotlin und Swift gewählt werden. Im Rahmen der Seminarfacharbeit fiel die Entscheidung auf Rust, da sich die Programmiersprache einerseits durch ihre hohe Geschwindigkeit und andererseits durch das starke Reduzieren unerwarteter Fehler auszeichnet. Hierzu gehört vor allem die optimierte Speicherverwaltung.

Aufgrund der genannten Vorteile fiel die Entscheidung auf Tauri. Im Allgemeinen ermöglichte Tauri, den Fokus der Arbeit auf die Implementierung der schnellen Suchfunktion zu legen. Es war beispielsweise für das Design des Programms bereits eine vorgefertigte Ordnerstruktur vorhanden, die lediglich noch anzupassen war. Somit war es möglich, den Fokus auf die Priorität der schnellen Suche zu legen, ohne dabei das Design zu vernachlässigen.

5.2 Auswahl einer geeigneten Datenbanktechnologie

Zur Auswahl einer geeigneten Datenbanktechnologie wurden die Technologien SQLite mit Rusqlite und SurrealDB in Betracht gezogen, wobei die endgültige Wahl auf Rusqlite fiel. Dies ist eine Erweiterung der Programmiersprache Rust, auch Crate genannt. Rusqlite baut auf SQLite, also einer Bibliothek mit einem relationellen Datenbanksystem, auf und bindet diese Technologie direkt in Rust ein. Außerdem erweitert Rusqlite den Semi-Explorer um zusätzliche Funktionen. Zu den hinzugefügten Funktionen zählen das Ausführen von SQL-Abfragen sowie das Erstellen und Verbinden mit SQLite-Datenbanken. Aufgrund der einfachen Implementierung in Rust wurde Rusqlite zum Speichern relevanter Informationen von Dateien genutzt.

Die Entscheidung gegen SurrealDB wurde aufgrund der als unzureichend empfundenen Dokumentation sowie der Nutzung einer eigenen Datenbanksprache namens SurrealQL getroffen. Zudem wurde SurrealDB als zu umfangreich für die Zwecke der Seminarfacharbeit erachtet, da es eine Vielzahl fortgeschrittener Funktionen umfasst, die für diese Seminarfacharbeit nicht relevant sind.

5.3 Evaluierung von Parallelisierungstechnologien

Wie bei Rusqlite handelt es sich bei Rayon um eine Erweiterung der Programmiersprache Rust. Rayon kann zur Parallelisierung von Operationen verwendet werden. Zu diesem Zweck stellt Rayon verschiedene Möglichkeiten bereit, darunter Threadpools und parallele Iteratoren. Zur Erklärung der beiden Begriffe: ein Threadpool ist ein Softwareentwurfsmuster, welches Threads verwaltet; ein Thread ist ein Ausführungsstrang. Ein Iterator ist ein Zeiger, mit welchem eine Menge durchlaufen werden kann. Ein wesentlicher Vorteil von Rayon liegt in seiner ressourcensparenden Funktionsweise und der Vermeidung von Unvorhersehbarkeiten, die bei der Parallelisierung von Prozessen auftreten können, wie zum Beispiel das Stillstehen zweier Threads, welche auf sich gegenseitig warten. Des Weiteren wurde Rayon verwendet, um die Datenverarbeitung im Backend zu parallelisieren, insbesondere bei der Suche und beim Füllen der Datenbank.

Für die Seminarfacharbeit wurde auch die Crate Tokio in Erwägung gezogen. Diese ist für Aufgaben ausgelegt, bei denen viele Threads genutzt werden, welche allerdings nicht dauerhaft arbeiten. Rayon hingegen ist für komplexe Berechnungen optimiert, bei denen die Threads permanent genutzt werden, was beim Durchsuchen und Erstellen der Datenbank unsere Intension ist.

6 Umsetzung des Dateiverwalters

6.1 Strukturierung des Programms

Die nachfolgende Abbildung 4 veranschaulicht die Implementierung des Programms in vereinfachter Form. Das Programm ist in vier Hauptabschnitte unterteilt. Die Pfeile kennzeichnen den Aufruf eines bestimmten Abschnitts. Der erste Abschnitt, der in der Abbildung rot dargestellt ist, verwaltet die Projektkonfiguration. Der Bereich, der sich mit der Datenbank und ihrer Verwaltung befasst, ist blau gekennzeichnet. Ein weiterer Teil des Programms widmet sich den verschiedenen Nutzerinteraktionen, wie beispielsweise der Anzeige von Dateien und dem Kontextmenü, welches beim Rechtsklick auf eine Datei erscheint. Dieser Teil ist gelb dargestellt. Die Nutzerinteraktionen interagieren ausschließlich mit dem letzten Abschnitt, dem Frontend, welcher grün dargestellt ist.

Die Initialisierung der Bereiche abgesehen von den weiteren Nutzerinteraktionen erfolgt durch den Programmstart. Dadurch wird die Projektkonfiguration geladen, welche der Nutzer in der Konfigurationsdatei anpassen kann. In den vorliegenden Konfigurationsdateien können unter anderem Dateierweiterungen, die nur bei der Suche angezeigt werden sollen, Suchfavoriten und der Kopiermodus konfiguriert werden. Die Projektkonfiguration ist demzufolge programmweit verfügbar. Des Weiteren wird die Datenbank erstellt, beziehungsweise aktualisiert und die ständige Aktualisierung während der Laufzeit gestartet. Daher wird ständig auf die Datenbank zugegriffen. Die Verwaltung dieser Operationen obliegt dem Manager. Zusätzlich werden im Manager benötigte Parameter bereitgestellt, bevor die eigentlichen Programmabschnitte aufgerufen werden. Der Grund dafür ist, dass die Programmabschnitte oftmals dieselben Parameter benötigen und es daher effizienter ist, sie zentral zu beschaffen. Das Frontend wird auch geladen.

Nach dem Abschluss der Initialisierung, werden sämtliche Operationen vom Frontend aus gestartet. Im Falle einer Suche nach einer Datei durch den Nutzer wird eine Anfrage an den Manager gesendet. Der Manager veranlasst daraufhin die Suche. Schließlich werden die Suchergebnisse an das Frontend zurückgesendet. Funktionen, abgesehen von der Suche, werden von den weiteren Nutzerinteraktionen übernommen.

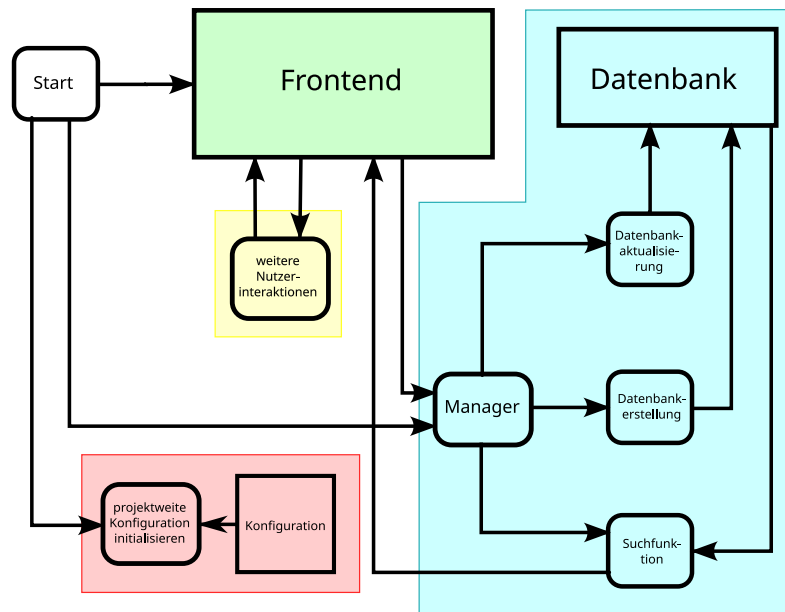


Abbildung 4: vereinfachter schematischer Aufbau des Programms [40]

6.2 Aufbau der Benutzeroberfläche und interne Kommunikation

Die Benutzeroberfläche setzt sich aus zwei Hauptbereichen sowie drei verborgenen Elementen zusammen. Im oberen Bereich der Benutzeroberfläche befinden sich zunächst drei Buttons, die es dem Nutzer ermöglichen, in andere Ordner zu gelangen. Das Programm verfügt zusätzlich neben dem „Vor- und Zurück“-Button über einen weiteren Button, mit welchem man in den Überordner gelangt. Des Weiteren gibt es zwei Eingabefelder für den Dateipfad und die Suchanfrage. Der Dateipfad besitzt einen entsprechenden „Go To“-Button. Darüber hinaus existiert ein Button in Form einer Lupe, der zur Bestätigung der Suchanfrage dient sowie ein Button zum Anzeigen der Suchoptionen.

In den Suchoptionen steht dem Nutzer ein Eingabefeld zur Verfügung, in dem er nach den Dateitypen filtern kann, die er angezeigt haben möchte. Darunter befinden sich Favoriten, die der Benutzer vorher individuell festlegen kann, die zusätzlich zur Filterung der Suche ausgewählt werden können. Beispielsweise legt der Nutzer als Favorit „Bilddateien“ fest. Wählt der Nutzer dieses Feld vor der Suche aus, werden ihm anschließend nur Bilddateien angezeigt. Zudem besteht die Möglichkeit, mittels eines entsprechenden Buttons das Eingabefeld wieder zurückzusetzen, sodass der Benutzer dieses erneut verwenden kann.

Im unteren Abschnitt befindet sich eine Tabelle, in welcher die Dateieinträge beziehungsweise Suchergebnisse präsentiert werden. Mittels Rechtsklick auf die Tabelle können zusätzliche Funktionen aufgerufen werden, wie das Kopieren, Einfügen, Ausschneiden, Umbenennen und Löschen einer Datei. Beim Umbenennen einer Datei öffnet sich ein neues Eingabefeld, in dem der Nutzer einen neuen Dateinamen eingeben kann.

Zur Optimierung der Benutzerfreundlichkeit des Semi-Explorers werden zwei Konfigurationsdateien verwendet. Beim Programmstart werden sie gelesen und deren Inhalt in Konstanten gespeichert. Diese können anschließend jederzeit von Programmteilen abgefragt werden. Darüber hinaus besteht die Möglichkeit, das Farbschema anzupassen.

Die Kommunikation zwischen Frontend und Backend wird von Tauri geregelt. Damit das Frontend Funktionen aufrufen kann, müssen diese zunächst als vom Frontend aufrufbare Funktionen deklariert werden. In der Folge ist es JavaScript möglich, derartige Funktionen unmittelbar aufzurufen.

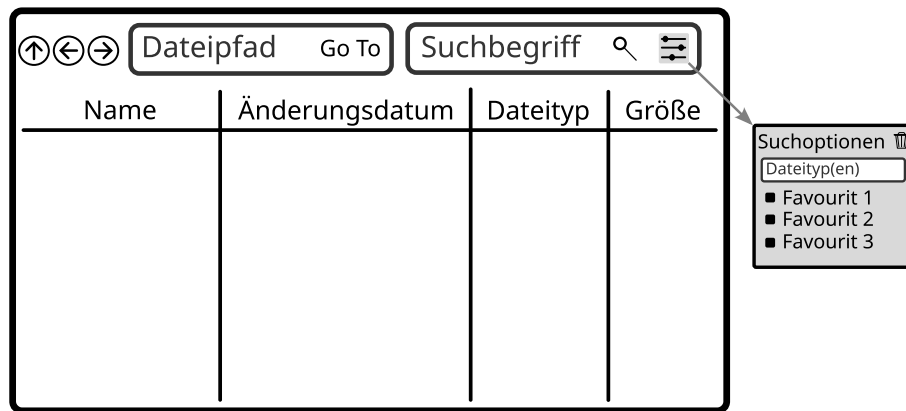


Abbildung 5: Benutzeroberfläche mit Suchoptionen [34]

6.3 Visualisierung des Dateisystems

Die Anzeige von Dateien und Ordnern in einem Verzeichnis erfolgt im Frontend. Der Nutzer hat die Möglichkeit, den „Go To“-Button anzuklicken. In der Folge wird der aktuell angegebene Dateipfad aus dem Eingabefeld ausgelesen. Im Anschluss wird der Dateipfad an das Backend übergeben und dessen Inhalt abgefragt. Tritt kein Fehler auf, werden die vorherigen Einträge gelöscht und mögliche Fehlermeldungen werden ausgeblendet.

Zunächst wird überprüft, ob es sich um einen validen Pfad handelt. Ist dieser Fall gegeben, werden alle Einträge des Verzeichnisses ausgelesen und weitere Informationen erfasst. Die Informationen umfassen den Dateinamen, die Dateiart, das letzte Änderungsdatum und die Größe der Datei. Zusätzlich wird der Pfad jedes Eintrags übermittelt. Nach Erhalt dieser Daten im Frontend wird für jeden Eintrag eine neue Zeile in der Tabelle erstellt. Anschließend wird die Zeile mit den weiteren Informationen gefüllt. Danach werden der Dateipfad und der Dateityp der entsprechenden Datei im Hintergrund mit in der Zeile gespeichert. Bei auftretenden Fehlern erfolgt eine entsprechende Fehlermeldung.

6.4 Realisierung der Datenbankgenerierung

Die Generierung der Datenbank erfolgt in zwei Schritten. Zunächst wird die Datenbank erstellt, falls sie noch nicht existiert. Anschließend werden die gewünschten Dateien, welche noch nicht in der Tabelle vorhanden sind, eingefügt. Hierbei ist es wichtig, dass die Datenbank dem Suchprozess möglichst viele Berechnungen abnimmt. Um dies zu erreichen, wird in der Datenbank der Name der Datei, der absolute Pfad, der Dateityp und die Bedeutung der Datei in Form einer Vektoreinbettung gespeichert. Die Verarbeitung der Dateien erfolgt parallel und in mehreren Chargen, um eine möglichst effiziente und schnelle Datenverarbeitung zu gewährleisten und die Datenbank schnell zu füllen.

Das Befüllen der Datenbank beginnt mit einer SQL-Abfrage, welche alle bereits vorhandenen Einträge beschafft. Im Anschluss wird das Dateisystem durchlaufen, wobei alle Dateien, die nicht bereits in der Datenbank vorhanden sind, zunächst gefiltert und dann in Chargen zur Verarbeitung weitergegeben werden. Im Zuge der Verarbeitung wird zunächst die Dateierweiterung vom Namen entfernt, sodass beim Vektoreinbetten ausschließlich der tatsächliche Dateiname verwendet wird. Dieser wird anschließend in Tokens aufgeteilt und in Indexe umgewandelt. Die erstellten Indexe werden anschließend genutzt, um die Einbettungen aus der Gewichtsmatrix auszulesen. Eine Gewichtsmatrix bezeichnet eine Ansammlung von Vektoren, deren Elemente die optimierten Gewichte sind. Die dadurch erzeugte Vektoreinbettung wird schließlich in einen Dateityp umgewandelt, der in der Datenbank gespeichert werden kann. Abschließend erfolgt die Eintragung des Namens, des Pfads, des Typs und der Vektoreinbettung in die Datenbank.

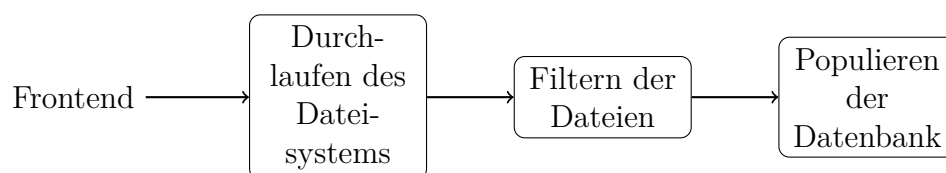


Abbildung 6: Vereinfachte Funktionsweise des Generierens der Datenbank [37]

6.5 Aktualisieren der Datenbank bei Dateisystemänderungen

Für eine präzise Suche ist eine stets aktuelle Datenbank erforderlich. Um diese zu gewährleisten, ohne permanent das gesamte Dateisystem zu durchsuchen und mit der Datenbank abzugleichen, wurde die Crate Notify genutzt.

Zu Beginn des Programms wird die Überwachung von Verzeichnissen, die der Nutzer zum Indizieren angibt, in einem separaten Thread initialisiert. Dadurch wird sichergestellt, dass Notify im Hintergrund weiterlaufen kann, ohne das Hauptprogramm zu beeinflussen. Das Programm reagiert auf eines der folgenden Ereignisse, indem es entsprechend handelt: Hinzufügen, Umbenennen oder Löschen. Anderenfalls befindet es sich im Ruhemodus und überwacht das Dateisystem lediglich.

Wird ein neuer Ordner oder eine neue Datei registriert, wird diese in die Datenbank eingefügt. Im Falle eines Ordners ist eine zusätzliche Durchsuchung des Inhalts erforderlich, da dieser nicht ohne Weiteres erkannt wird.

Im Rahmen einer Umbenennung wird der ursprüngliche Pfad aus der Datenbank entfernt und der neue Pfad hinzugefügt. Bei Ordnern ist aufgrund der fehlenden Mitteilung des ehemaligen Pfads wiederum eine separate Handlung erforderlich. Um dieses Problem zu lösen, wird der Inhalt des Ordners, in dem sich der umbenannte Ordner befindet, mit der Datenbank verglichen. Im Anschluss erfolgt die Entfernung sämtlicher nicht mehr existierender Elemente sowie die Hinzufügung neuer Elemente. Dadurch kann der ehemalige Ordnername erfolgreich entfernt werden.

Beim Löschen einer Datei wird der entsprechende Pfad aus der Datenbank entfernt. Das Löschen eines Ordners ist in diesem Fall nicht möglich, da dieses Ereignis aus bisher noch unerklärlichen Gründen nicht erkannt wird. Trotz intensiver Recherche und einer Vielzahl von Tests war es nicht möglich, das Problem zu lösen. Demzufolge verbleibt ein Ordner in der Datenbank, auch wenn er eigentlich nicht mehr existiert. Dies kann dazu führen, dass dem Benutzer unter den Suchergebnissen Ordner angezeigt werden, die nicht mehr vorhanden sind. Die vorliegende Problematik konnte nur in Teilen durch die direkte Verknüpfung der Löschfunktion mit der Datenbank behoben werden. Dies hat zur Folge, dass vom Benutzer gelöschte Ordner direkt entfernt werden. Sollten Ordner durch Aktionen außerhalb unseres Programms gelöscht werden, ist eine Nachverfolgung dieser Vorfälle jedoch nicht möglich. Lediglich ein Neustart des Programms würde zu einer vollständigen Aktualisierung der Datenbank führen.

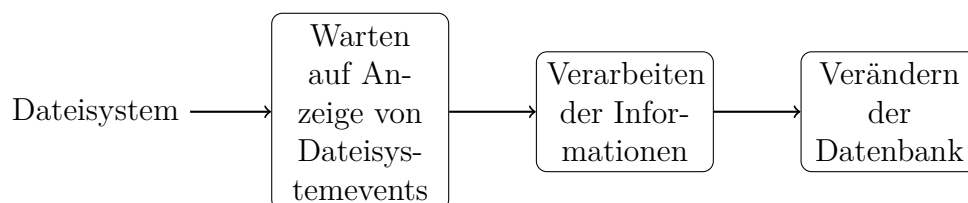


Abbildung 7: Vereinfachte Funktionsweise der Aktualisierung [38]

6.6 Verwirklichen der Dateisuche

Die Suchfunktion des Semi-Explorers wurde mit dem Ziel konzipiert, eine hohe Leistungsfähigkeit und Modularität zu gewährleisten sowie die relevantesten Ergebnisse zu ermitteln. Um die genannten Funktionalitäten zu ermöglichen, wurde eine spezifische Zwischenfunktion genutzt. Die Zwischenfunktion übernimmt die Kommunikation mit dem Frontend und stellt einige Argumente für die eigentliche Suchfunktion bereit. Dies sorgt dafür, dass die Suchfunktion einfach zu integrieren bleibt. Der eigentliche Suchalgorithmus beginnt mit der Verarbeitung der Daten, welche die Datenbank bereitstellt. Dieser Prozess erfolgt parallel zum Erhalt der Daten. Bei dieser Vorgehensweise erfolgt die Verwendung einer SQL-Abfrage, welche nur mögliche Einträge ausgibt, indem der Pfad und der Dateityp bereits in die Abfrage integriert werden. Die Pfade und Dateitypen sind bereits indiziert, um so wenig Zeit wie möglich für die Beschaffung der Daten zu beanspruchen.

Im darauffolgenden Schritt des Algorithmus' werden lediglich die Pfade und die Vektoreinbettungen des Semi-Modells genutzt. Im ersten Schritt erfolgt die Bestimmung der Dateinamen mithilfe der Pfade. Im Anschluss daran erfolgt die Berechnung der Levenshtein-Ähnlichkeiten zwischen den Dateinamen und der Suchanfrage. Daraufhin werden die Ergebnisse mit höchster Levenshtein-Ähnlichkeit zu einem Datentyp transformiert, der von dem Frontend genutzt werden kann. Dann werden die Ergebnisse an das Frontend gesendet. Anschließend beginnt die Suche mit den Vektoreinbettungen. Dazu wird zunächst die eingegebene Suchanfrage in einen Vektor eingebettet und dann die Kosinus-Ähnlichkeit zwischen den Vektoreinbettungen der Dateien und der Suchanfrage berechnet. Dieser Vorgang findet in einem Threadpool statt, um die Kosinus-Ähnlichkeit für mehrere Einträge gleichzeitig zu berechnen. Daraufhin erfolgt das Auswählen der besten Einträge basierend auf ihrer Kosinus-Ähnlichkeit zur Suchanfrage. Letztendlich werden die ausgewählten Einträge ebenfalls umgewandelt und an das Frontend weitergeleitet.

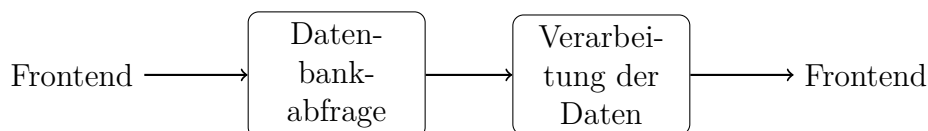


Abbildung 8: Vereinfachte Funktionsweise des Suchalgorithmus' [39]

6.7 Essentielle Funktionen für den Nutzer

Ein File Explorer muss neben der Funktion, Dateien anzuzeigen und Ordner zu durchsuchen, über weitere unerlässliche Funktionen verfügen. Zu diesen zählen das Kopieren und Einfügen von Dateien, das Umbenennen, das Löschen sowie das Öffnen dieser. Es besteht zudem die Option, Dateien auszuschneiden. Dieser Prozess vereint sowohl die Kopier- als auch die Löschfunktion. Im Rahmen der Implementation wurde für jede dieser Aktionen eine Funktion erstellt, einschließlich weiterer Zusatzfunktionen zur Bereinigung des Dateipfads. Beim Aufruf einer dieser Funktionen wird dem Frontend ein Dateipfad übermittelt. Der Dateipfad dient als Referenz zur Datei, mit der interagiert wird.

Die Kopierfunktion bietet zwei verschiedene Modi zum Kopieren an: „Clipboard“ und „File“. Je nach Modi erfolgt entweder das Kopieren des Dateiinhalts in die Zwischenablage mit einer Speicherung des Dateinamens in einer Datei oder das direkte temporäre Kopieren der Datei in einen Ordner zusammen mit der Speicherung des Dateinamens in einer beiliegenden Datei. Ordner werden immer im „File“ Modus kopiert.

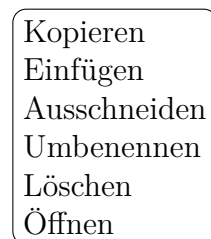


Abbildung 9: Vereinfachte Anordnung des Kontextmenüs [41]

Das Einfügen erfolgt durch die Ausführung des entsprechenden Gegenstücks. Zunächst wird der Kopiermodus abgefragt, bevor entsprechend gehandelt wird. Im Anschluss wird der Inhalt entweder aus der Zwischenablage ausgelesen, eine Datei am Zielpfad erstellt und der Inhalt anschließend eingefügt oder die temporär gespeicherte Datei wird zum Zielpfad kopiert. Der Kopiermodus wird von der Konfigurationsdatei eingelesen und ist daher vom Benutzer beliebig anpassbar.

Die Umbenennungsfunktion ist die einzige dieser Funktionen, die auch den neuen Dateinamen übergeben bekommt. Im Anschluss erfolgt die Umbenennung der Datei, sofern diese Bezeichnung noch nicht vergeben ist.

Die Lösch- und Öffnungsfunktion hingegen rufen direkt die Funktionen des Betriebssystems auf. Die entsprechenden Funktionen zum Kopieren und Umbenennen einer Datei werden ebenfalls vom Betriebssystem bereitgestellt.

7 Vergleich des Semi-Explorers mit dem Windows-Explorer

Als Gemeinsamkeit ist festzuhalten, dass es sich sowohl bei dem Semi-Explorer als auch beim Windows-Explorer um ein Mittel zur Anzeige des Dateisystems handelt. Damit sind die grundlegenden Funktionen, die einer solcher Explorer benötigt, in beiden vorhanden. Zu den entsprechenden Funktionen gehören das Anzeigen sowie das Kopieren, Löschen, Umbenennen und Öffnen von Dateien.

Der erste Unterschied zeigt sich in der Funktionsvielfalt des Windows-Explorers, der einen deutlich umfangreicheren Funktionsumfang aufweist. Anzumerken ist jedoch, dass beim Semi-Explorer das Hauptaugenmerk nicht auf diesem spezifischen Kriterium lag.

Der Semi-Explorer weist im Gegensatz jedoch Funktionen auf, die im Windows-Explorer nicht verfügbar sind. So kann durch den Nutzer bestimmt werden, wie viel Leistung dieser maximal in Anspruch nehmen darf. Im inaktiven Zustand wird trotz der Anpassung nur minimal Leistung verbraucht. Unsere Erfahrung zeigt, dass der Windows-Explorer dies teilweise nicht gewährleisten kann und mehr Leistung in Anspruch nimmt.

Außerdem sind die Suchergebnisse verschieden. Beim Windows-Explorer sind diese meist nur direkte Übereinstimmungen. Der Semi-Explorer zeigt hingegen auch ähnliche Ergebnisse an und findet Dateinamen, die nicht ähnlich geschrieben sind, jedoch eine sinnhafte Verknüpfung aufweisen. Wird demzufolge bei der Suche ein Rechtschreibfehler gemacht, führt dies zu einer starken Reduktion der gefundenen Ergebnisse bei Windows. So zeigte die Erfahrung, dass der Windows-Explorer bei einem einzigen Fremdzeichen bereits keine Ergebnisse mehr liefern kann. Im Gegensatz gibt der Semi-Explorer trotz eines möglichen Rechtschreibfehlers valide Ergebnisse zurück.

Durch eine präzise Einschränkung der Suchergebnisse durch den Nutzer, werden die Ergebnisse des Semi-Explorers zudem noch treffender. Diese Funktionalität ist beim Windows File Explorer lediglich in eingeschränktem Maße gegeben.

Die nachfolgende Abbildung 10 veranschaulicht den Vergleich zwischen den Suchzeiten des im Rahmen der Seminarfacharbeit implementierten Suchalgorithmus' und den Suchzeiten des Windows 11 File Explorers. Hierbei wird nochmals zwischen der Suche ausschließlich mit der Levenshtein-Distanz und mit der Levenshtein-Distanz sowie dem Semi-Modell unterschieden. Zu diesem Zweck wurden insgesamt 225 000 Dateien auf ihren Namen durchsucht, die zuvor indiziert wurden. Es konnte festgestellt werden, dass die Suchzeit des Semi-Explorers die des Windows-Explorers um mehr als das Doppelte übertrifft. Obwohl das Semi-Modell zusätzliche 50 Millisekunden benötigt, ist die Suche immer noch doppelt so schnell. Wie bereits festgestellt, konnte der Windows-Explorer bei der Abänderung des Dateinamens um ein Zeichen keine Ergebnisse mehr liefern. Der Semi-Explorer hingegen findet die Datei immer noch, selbst nach einer Abänderung von fünf Zeichen. Insgesamt benötigt der Semi-Explorer demzufolge nicht nur halb so viel Zeit wie der Windows-Explorer zum Suchen, sondern findet zusätzlich mehr Ergebnisse.

Zur Zeitmessung wurde zunächst derselbe Ordner sowohl im Windows-Explorer als auch im Semi-Explorer indiziert. Nachdem beide Datenbanken vollständig erstellt wurden, begann die Suche und die Aufzeichnung. Bei der Messung wurde auf eine sehr ähnliche Auslastung des Computers geachtet. Trotzdem stimmt die Auslastung nie vollständig überein, wodurch Messungenauigkeiten entstanden sein könnten. Diese sind jedoch aufgrund ihrer kleinen Abweichung vernachlässigbar. Die Suche erfolgte jeweils fünfmal mit demselben Suchbegriff, um die Messungenauigkeit noch weiter zu verringern. Danach wurde mithilfe eines Programms zum Bearbeiten von Videos die Suchzeit auf eine sechzigstel Sekunde genau ermittelt und der Durchschnitt der fünf Suchzeiten gebildet.

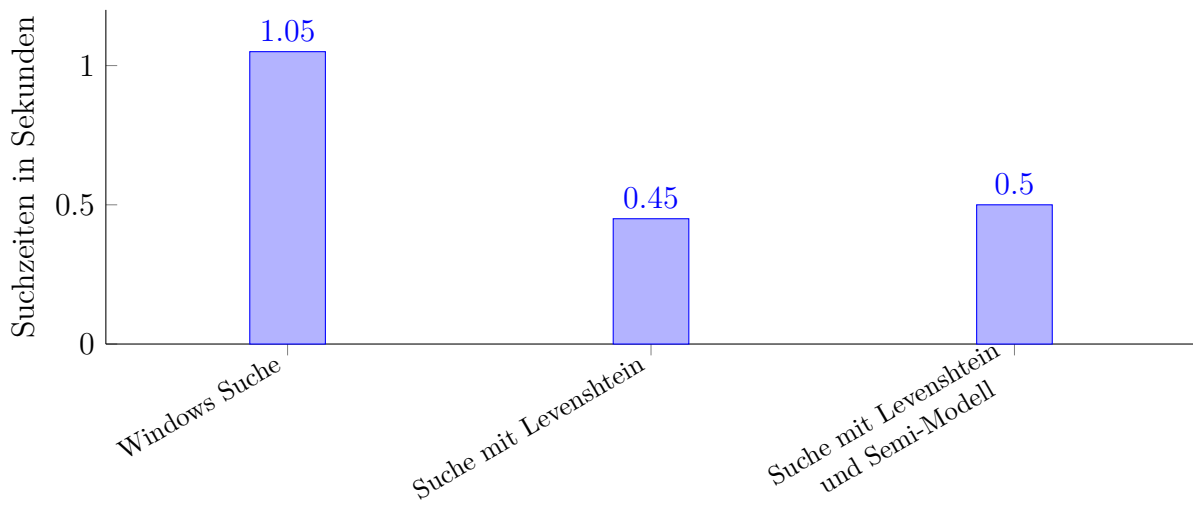


Abbildung 10: Diagramm zum Vergleich von Suchzeiten unseres Algorithmus' mit dem Windows 11 File Explorer [36]

8 Zusammenfassung

Abschließend kann festgehalten werden, dass die Nutzung von elektronischen Medien im derzeitigen Internet-Zeitalter unerlässlich geworden ist. Für Suchvorgänge auf Festplatten privater und dienstlicher Computer wird viel Zeit aufgewendet. Außerdem zeigt der Windows 11-Explorer bei Rechtschreibfehlern keine oder falsche Dateien an. Die begrenzte Ressource Zeit wird verschwendet und die Suche der Dateien muss wiederholt werden.

Wir nahmen uns vor, mit dieser Seminarfacharbeit einen Beitrag zur Verbesserung der Funktionalität von File Explorern zu leisten. Durch eine intensive Auseinandersetzung gelang es uns auch, dies in unserem Eigenanteil umzusetzen. Wir entwickelten einen eigenen File Explorer, den Semi-Explorer, mit schneller Suchfunktion. Mit diesem konnten wir eine deutliche Steigerung der Suchgeschwindigkeit erzielen, wodurch wir unser zu Beginn gesetztes Ziel erreichen konnten.

In der Gegenüberstellung der Testwerte zeigt sich, dass der von uns entwickelte Explorer eine 2,3-mal höhere Verarbeitungsgeschwindigkeit aufweist, als der Windows 11-Explorer. Während der Windows-Explorer für eine Suche 1,05 Sekunden benötigt, beträgt die Suchzeit des Semi-Explorers lediglich 0,45 Sekunden. Im Vergleich zu den Ergebnissen der Umfrage des Startups amberSearch würden Mitarbeiter bei einem allein durch den Explorer verursachten Zeitverlust beim Gebrauch unseres Semi-Explorers lediglich 13 Minuten anstatt einer halben Stunde [2] für die Suche im Arbeitsalltag benötigen. Bei einer Betrachtung der Suchzeit über das gesamte Jahr mit einer Anzahl von circa 230 Arbeitstagen [2] würde sich diese demzufolge um drei Tage reduzieren und ausschließlich zwei Tage in Anspruch nehmen.

Der Semi-Explorer beinhaltet alle wesentlichen Funktionen, die für einen Nutzer relevant sind. Zu den Funktionen gehören das Kopieren, Ausschneiden, Umbenennen sowie das Speichern in unterschiedlichen Ordnern. Zusätzlich besteht für den Benutzer die Option, den File Explorer nach individuellen Vorlieben zu gestalten, indem er die Farbe anpasst. Die Dateisuche unseres Semi-Explorers ist außerdem anders gestaltet als bei herkömmlichen Explorern. Unser Semi-Explorer ermöglicht es, nach spezifischen Dateitypen zu suchen, die der Nutzer im Voraus angeben kann. Dadurch kann eine noch schnellere und effektivere Suche gewährleistet werden.

Vor der Arbeitsphase legten wir einen Zeitplan zur Fertigstellung der einzelnen Kapitel und des Programms fest. Einen Großteil unserer Zeit investierten wir, um uns die Programmiersprachen anzueignen, diese anzuwenden und auftretende Probleme zu lösen. Die Vorbereitung und Umsetzung des Programms erwies sich als äußerst zeitintensiv. Durch interne Arbeitsteilung, kontinuierliches Schreiben und Prüfen der Kapitel gelang es uns, den Zeitplan sowohl für den praktischen als auch theoretischen Teil einzuhalten.

Zukünftige Seminarfachgruppen könnten zum einen unser bereits bestehendes neuronales Netz und dessen Implementation optimieren, um die Suchgeschwindigkeit noch weiter zu steigern. Des Weiteren könnte ein neues neuronales Netz entwickelt werden, das in der Lage ist, den Inhalt von Dateien zu durchsuchen. Durch die Implementierung dieser Funktion wäre es möglich, Dateien nicht nur anhand ihrer Namen, sondern auch anhand ihres Inhalts zu finden. Ebenso könnte man sich weiterhin mit der Frage beschäftigen, inwiefern man den Semi-Explorer noch nutzerfreundlicher gestalten kann.

9 Literatur- und Quellenverzeichnis

Literaturverzeichnis

- [1] Zhang, Aston u. a. *Dive into Deep Learning*. Cambridge University Press, 2023. URL: <https://D2L.ai>.

Quellenverzeichnis

- [2] amberSearch. *Wie viel Zeit verbringen MitarbeiterInnen mit der unternehmensinternen Suche?* 13. Feb. 2024. URL: <https://ambersearch.de/roi-von-ki-tools/> (besucht am 05.05.2025).
- [3] Assecor. *Was ist SQL und eine SQL-Datenbank?* URL: <https://www.assecor.de/glossar/sql-datenbank> (besucht am 10.03.2025).
- [4] Autorenkollektiv. *Kreuzentropie*. 16. Dez. 2024. URL: <https://de.wikipedia.org/wiki/Kreuzentropie> (besucht am 28.04.2025).
- [5] Autorenkollektiv. *Levenshtein-Distanz*. 17. Jan. 2025. URL: <https://de.wikipedia.org/wiki/Levenshtein-Distanz> (besucht am 09.04.2025).
- [6] Autorenkollektiv. *One-hot*. 28. März 2025. URL: <https://en.wikipedia.org/wiki/One-hot> (besucht am 25.04.2025).
- [7] Autorenkollektiv. *Regulärer Ausdruck*. 13. Dez. 2024. URL: https://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck (besucht am 28.04.2025).
- [8] Autorenkollektiv. *Verlustfunktion*. 10. Dez. 2024. URL: [https://de.wikipedia.org/wiki/Verlustfunktion_\(Statistik\)](https://de.wikipedia.org/wiki/Verlustfunktion_(Statistik)) (besucht am 28.04.2025).
- [9] Autorenkollektiv. *Word2Vec*. 29. Apr. 2025. URL: <https://en.wikipedia.org/wiki/Word2vec> (besucht am 05.05.2025).
- [10] Autorenkollektiv. *Worteinbettung*. 6. Apr. 2025. URL: <https://de.wikipedia.org/wiki/Worteinbettung> (besucht am 25.04.2025).
- [11] Bonander, Austin. *sqlx*. 7. Mai 2019. URL: <https://crates.io/crates/sqlx> (besucht am 04.09.2025).
- [12] Canadia. *Einfach erklärt: Was ist ein Frontend?* URL: <https://canida.io/wissen/frontend> (besucht am 10.03.2025).
- [13] Candia. *Canida - Einfach erklärt: Was ist ein Backend?* URL: <https://canida.io/wissen/backend> (besucht am 10.03.2025).
- [14] ESM-Computer. *Aufgaben eines Dateisystems*. URL: <https://www.esm-computer.de/ratgeber-infos/lexikon/dateisystem/> (besucht am 17.01.2025).
- [15] Foundation, Tauri. *Tauri 2.0*. URL: v2.tauri.app (besucht am 10.03.2025).
- [16] Franz, Sonja. *Wie Suchalgorithmen funktionieren: Die Levenshtein-Distanz*. 22. Feb. 2018. URL: <https://www.integer-net.de/blog/wie-suchalgorithmen-funktionieren-levenshtein-distanz> (besucht am 08.04.2025).
- [17] Geißler, Otto. *Warum ist Rust so beliebt?* 27. März 2024. URL: <https://www.datacenter-insider.de/programmiersprache-rust-vorteile-entwickler-acc3df973e40b2f0cb630c712f779224f/> (besucht am 29.04.2025).

- [18] Group, Online Soultions. *Was ist der Levenshtein-Algorithmus?* URL: <https://www.onlinesolutionsgroup.de/blog/glossar/1/levenshtein-algorithmus/> (besucht am 09.04.2025).
- [19] Group, PostgreSQL Global Development. *postgresql*. URL: <https://www.postgresql.org/> (besucht am 04.09.2025).
- [20] gwenn. *rusqlite*. 21. Nov. 2014. URL: <https://crates.io/crates/rusqlite> (besucht am 10.03.2025).
- [21] IBM. *Maschinelles Lernen*. URL: <https://www.ibm.com/think/topics/machine-learning> (besucht am 25.04.2025).
- [22] Kingma, Diederik und Ba, Jimmy. *Adaptive Moment Estimation*. 2015. URL: <https://arxiv.org/pdf/1412.6980> (besucht am 28.04.2025).
- [23] Landwehr, Jesko. *Was ist ein Framework? -Definition und Erklärung*. 20. Nov. 2020. URL: <https://www.it-talents.de/it-wissen/framework/> (besucht am 09.04.2025).
- [24] Leipzig, Universität. *Wortschatz Leipzig*. URL: <https://wortschatz.uni-leipzig.de/en/download/English> (besucht am 27.04.2025).
- [25] Lenovo. *Was ist der Datei-Explorer?* URL: <https://www.lenovo.com/de/de/glossary/file-explorer/?orgRef=https%253A%252F%252Fwww.google.com%252F&srsltid=AfmB0ooMuliihfjtL5MA7GI1DYmbQWDPeK7JXOT9nIomZSblEhbfvB1D> (besucht am 17.01.2025).
- [26] Ltd., SurrealDB. *surrealdb*. URL: <https://surrealdb.com/> (besucht am 04.09.2025).
- [27] Mazare, Laurent. *tch*. 27. Feb. 2019. URL: <https://crates.io/crates/tch> (besucht am 04.09.2025).
- [28] Peters, Marcel. *Was ist ein Explorer?* 9. Sep. 2018. URL: https://praxistipps.chip.de/was-ist-ein-explorer-ei%20nfach-erklaert_41438 (besucht am 17.01.2025).
- [29] Solutions, BANK MEDIA Software. *Zeit ist Geld*. 30. März 2021. URL: <https://www.bankmedia.de/blog/zeit-ist-geld-und-wie-finanzinstitute-dauerhafte-ressourcenknappheit-in-der-it-vermeiden-koennten/> (besucht am 05.05.2025).
- [30] Sqlite. *sqlite*. 9. Mai 2000. URL: <https://sqlite.org/> (besucht am 04.09.2025).
- [31] Stone, Josh. *rayon*. 10. Dez. 2015. URL: <https://crates.io/crates/rayon> (besucht am 10.03.2025).
- [32] Studyflix. *Funktionen einer Datenbank*. URL: <https://studyflix.de/informatik/datenbank-7460> (besucht am 10.03.2025).
- [33] Talend. *Datenintegrität: Bedeutung, Arten und Risiken*. URL: <https://www.talend.com/de/resources/was-ist-datenintegritaet/%20#:~:text=Der%20Begriff%20Datenintegrit%C3%A4t%20bezieht%20sich,Daten%20fallen%20unter%20diesen%20Begriff.> (besucht am 10.03.2025).

Abbildungsverzeichnis

- [34] Fischer, Nino, Nolle, Jessica und Schultheis, Magnus. *Benutzeroberfläche*. Screenshot des Programms. 24. Apr. 2025.

- [35] Fischer, Nino, Nolle, Jessica und Schultheis, Magnus. *Diagramm zum Vergleich von Suchzeiten mit Windows 10*. eigene Darstellung. 24. Feb. 2025.
- [36] Fischer, Nino, Nolle, Jessica und Schultheis, Magnus. *Diagramm zum Vergleich von Suchzeiten mit Windows 11*. eigene Darstellung. 24. Feb. 2025.
- [37] Fischer, Nino, Nolle, Jessica und Schultheis, Magnus. *Funktionsweise der Generierung der Datenbank*. eigene Darstellung. 15. März 2025.
- [38] Fischer, Nino, Nolle, Jessica und Schultheis, Magnus. *Funktionsweise des Aktualisieren der Datenbank*. eigene Darstellung. 24. Apr. 2025.
- [39] Fischer, Nino, Nolle, Jessica und Schultheis, Magnus. *Funktionsweise des Suchalgorithmus*. eigene Darstellung. 15. März 2025.
- [40] Fischer, Nino, Nolle, Jessica und Schultheis, Magnus. *vereinfachter Aufbau des Programms*. eigene Darstellung. 26. Apr. 2025.
- [41] Fischer, Nino, Nolle, Jessica und Schultheis, Magnus. *Visualisierung des Kontext Menüs*. eigene Darstellung. 30. Apr. 2025.
- [42] Fischer, Nino, Nolle, Jessica und Schultheis, Magnus. *Visualisierung Worteinbettungen*. eigene Darstellung. 28. Apr. 2025.
- [43] Zhang, Aston u. a. *Continuous Bag of Words model (CBOW)*. 6. Sep. 2024. URL: [https://en.wikipedia.org/wiki/File:Continuous_Bag_of_Words_model_\(CBOW\).svg](https://en.wikipedia.org/wiki/File:Continuous_Bag_of_Words_model_(CBOW).svg) (besucht am 22.04.2025).
- [44] Zhang, Aston u. a. *Skip-gram*. 6. Sep. 2024. URL: <https://en.wikipedia.org/wiki/File:Skip-gram.svg> (besucht am 22.04.2025).

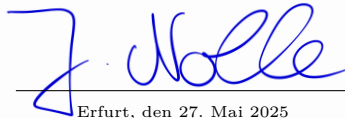
Eidesstattliche Erklärung

Hiermit erklären wir an Eides statt, dass wir die vorliegende Arbeit eigenständig und nur unter Verwendung der angegebenen Quellen angefertigt haben.



Erfurt, den 27. Mai 2025

Nino Fischer



Erfurt, den 27. Mai 2025

Jessica Nolle



Erfurt, den 27. Mai 2025

Magnus Schultheis