5. Word Embedding

5.1 Word Embedding

In [1]:
```
import pandas as pd
import re
import numpy as np
from gensim.models import Word2Vec
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import plotly.graph_objects as go
import ast
from sklearn.feature_extraction.text import TfidfVectorizer
import plotly.express as px
from nltk.corpus import stopwords
from sklearn.metrics.pairwise import cosine_similarity
```

In [2]:
```
df = pd.read_csv("data/clean/data.csv")


df = df[df["language_cld3"] == "en"]

if isinstance(df["tokens"].iloc[0], str):
    df["tokens"] = df["tokens"].apply(ast.literal_eval)

sentences = df["tokens"].dropna().tolist()


model = Word2Vec(
    sentences=sentences,
    vector_size=50,
    window=5,
    min_count=2,
    workers=4,
```

```
    epochs=100
)


print("Model trained!")
print("Vocabulary size:", len(model.wv))
print("Vector size:", model.wv.vector_size)


word = "love"
if word in model.wv:
    print(f"\nVector for '{word}':", model.wv[word][:10])
    print("\nMost similar words to 'love':")
    print(model.wv.most_similar(word, topn=5))
else:
    print(f"'{word}' not in vocabulary.")
```

```
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
```

```
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
```

```
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Exception ignored in: 'gensim.models.word2vec_inner.our_dot_float'
Model trained!
Vocabulary size: 65504
Vector size: 50

Vector for 'love': [ 1.084971   -0.7270534   2.1659267  -1.7042322   2.7113059   0.24227719
 -2.1262593  -6.229046   -2.8657782   1.9509404 ]

Most similar words to 'love':
[('loving', 0.7831743359565735), ('baby', 0.7558361887931824), ('cause', 0.6966375708580017), ('lov', 0.694631636142
7307), ('know', 0.6943686008453369)]
```

Diese Notiz beschreibt, wie die CSV-Datei **`data.csv`** eingelesen, gefiltert und für das Training eines **Word2Vec-Modells** (Gensim) aufbereitet wird. Anschliessend werden die gewählten Hyperparameter begründet.

Einlesen & Filtern der Daten

Datei: `data.csv`

Relevante Spalten:

- `language_cld3` — erkannte Sprache (typisch ISO-Code, z. B. `en` für Englisch)
- `tokens` — Liste der Token (Wörter) der Lyrics pro Zeile

Schritte:

1. CSV-Datei einlesen.
2. Alle Zeilen **behalten**, bei denen die erkannte Sprache **Englisch** ist (`language_cld3 == "en"` , Groß-/Kleinschreibung ignorieren).
3. Fehlende Werte (**NaN**) entfernen – insbesondere in der Spalte `tokens` .
4. Aus der Spalte `tokens` eine **Python-Liste von Listen** erzeugen (jede Zeile → eine Tokenliste).

> **Hinweis:** Für Word2Vec müssen es **Listen von Strings** sein.

Training des Word2Vec-Modells

Ziel: Lernen von Wortvektoren aus den Lyrics-Token. **Bibliothek:** `gensim.models.Word2Vec`

Wichtige Parameter

Parameter	Bedeutung
`sentences`	Eingabedaten als Liste von Wortlisten (siehe Schritt 1)
`vector_size=50`	Dimension der Wortvektoren (kompakte Darstellung, schnelleres Training; ausreichend für einen ersten Durchlauf)
`window=5`	Kontextfenster: betrachtet bis zu 5 Wörter **links und rechts** eines Zielworts → balanciert lokalen und mittleren Kontext
`min_count=2`	Ignoriert seltene Wörter (< 2 Vorkommen) → reduziert Rauschen & Vokabulargröße
`workers=4`	Nutzt 4 Threads zur Parallelisierung → schnelleres Training (abhängig von CPU)
`epochs=100`	100 Epochen (vollständige Durchläufe über die Daten) → stabilere Vektoren bei kleineren Datensätzen

Ergebnis

Ein trainiertes **Word2Vec-Modell**, das **semantische Beziehungen zwischen Wörtern** in einem **50-dimensionalen Raum** abbildet.

Beispiele:

- **Ähnlichkeiten** zwischen Wörtern (`model.wv.most_similar("love")`)

In [3]:
```
test_word = "love"
if test_word in model.wv:
    print(f"\nMost similar to '{test_word}':")
    for w, s in model.wv.most_similar(test_word, topn=5):
        print(f"  {w:15s} {s:.3f}")
else:
    print(f"'{test_word}' not in vocabulary.")
```

```
Most similar to 'love':
  loving          0.783
  baby            0.756
  cause           0.697
  lov             0.695
  know            0.694
```

Überprüfung eines Beispielwortes im Word2Vec-Vokabular

Nach dem Training kann überprüft werden, ob ein bestimmtes Wort hier **"love"** im Vokabular des trainierten Modells enthalten ist.

Schritte:

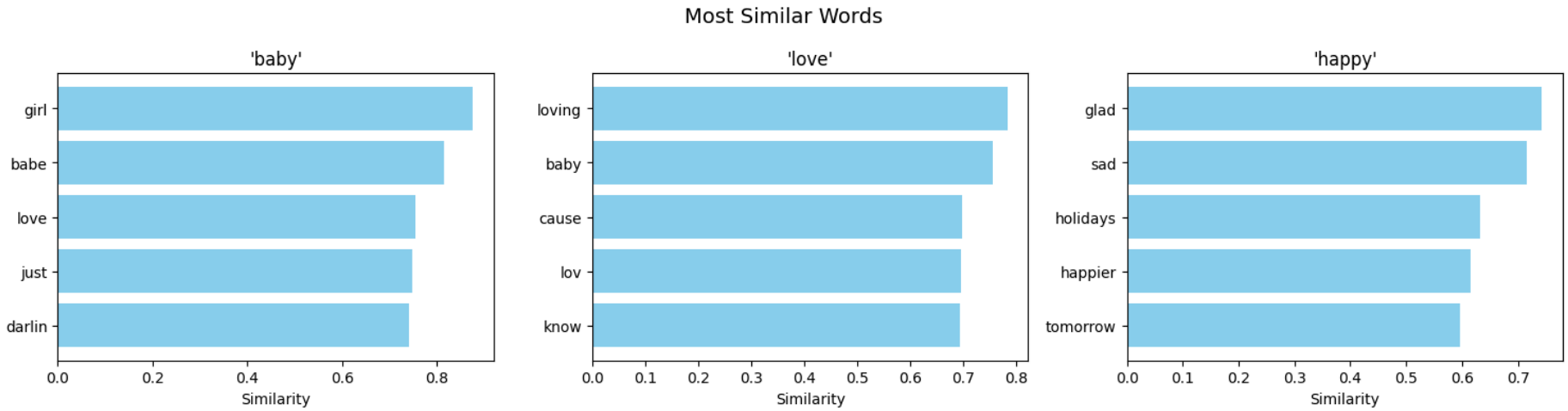
1. **Prüfen**, ob das Wort `"love"` im Vokabular vorhanden ist (`if word in model.wv:`).
2. **Falls ja:**
   - Gibt die **ersten 10 Werte** des zugehörigen Wortvektors aus (`model.wv[word][:10]`).
   - Zeigt die **5 ähnlichsten Wörter** basierend auf der Kosinusähnlichkeit (`model.wv.most_similar(word, topn=5)`).
3. **Falls nein:**
   - Gibt eine **Hinweismeldung** aus, dass das Wort nicht im Vokabular enthalten ist.

> 💡 **Hinweis:** Ob ein Wort im Vokabular vorhanden ist, hängt von der Häufigkeit im Trainingskorpus ab. Wörter mit weniger als `min_count` (hier 2) Vorkommen werden **nicht** im Modell gespeichert.

In [4]:

```python
def find_similar_words(word, model, top_n=5):
    if word not in model.wv:
        return None, None
    similar = model.wv.most_similar(word, topn=top_n)
    words = [w for w, _ in similar]
    scores = [s for _, s in similar]
    return words, scores


test_words = ["baby", "love", "happy"]

fig, axes = plt.subplots(1, len(test_words), figsize=(5 * len(test_words), 4))

if len(test_words) == 1:
    axes = [axes]

for ax, word in zip(axes, test_words):
    print(f"\n🔍 Finding words similar to '{word}'...")
    words, scores = find_similar_words(word, model, top_n=5)

    if words:
        for w, s in zip(words, scores):
            bar = '█' * int(s * 10)
            print(f"  {w:10} {bar} {s:.2f}")

        ax.barh(words, scores, color="skyblue")
        ax.set_title(f"'{word}'")
        ax.set_xlabel("Similarity")
        ax.invert_yaxis()
    else:
        ax.set_visible(False)
        print(f"  '{word}' not in vocabulary.")

fig.suptitle("Most Similar Words", fontsize=14)
fig.tight_layout()
plt.show()
```

```
🔍 Finding words similar to 'baby'...
  girl       ████████ 0.88
  babe       ████████ 0.81
  love       ███████ 0.76
  just       ███████ 0.75
  darlin     ███████ 0.74

🔍 Finding words similar to 'love'...
  loving     ███████ 0.78
  baby       ███████ 0.76
  cause      ██████ 0.70
  lov        ██████ 0.69
  know       ██████ 0.69

🔍 Finding words similar to 'happy'...
  glad       ███████ 0.74
  sad        ███████ 0.72
  holidays   ██████ 0.63
  happier    ██████ 0.62
  tomorrow   ██████ 0.60
```



Ähnlichkeitssuche im Word2Vec-Modell

Diese Funktion sucht nach den **ähnlichsten Wörtern** zu einem gegebenen Wort im Word2Vec-Modell. Sie gibt zwei Listen zurück:

- `words` → die ähnlichsten Wörter
- `scores` → die Ähnlichkeitswerte (Kosinus-Ähnlichkeit, zwischen 0 und 1)

Wenn das Wort **nicht im Vokabular** vorhanden ist, wird `(None, None)` zurückgegeben.

Beispielerklärungen

`happy – sad`

→ Überraschend ähnlich, obwohl es Gegenteile sind. **Warum?** Word2Vec versteht **Kontextähnlichkeit**, nicht logische Gegensätze. Beide Wörter treten oft in ähnlichen Satzstrukturen auf:

> "I feel happy today." / "I feel sad today." Daher liegen sie **räumlich nah**, obwohl sie **semantisch gegensätzlich** sind.

`happy – happier`

→ Grammatische Variante desselben Wortstamms („happy" → „happier"). Das Modell erkennt **Formverwandtschaften**.

In [5]:
```python
def explore_similar_words(word, model, top_n=10):

    if word not in model.wv:
        raise ValueError(f"'{word}' not in model vocabulary.")


    similar = model.wv.most_similar(word, topn=top_n)
    words = [word] + [w for w, _ in similar]
    scores = [1.0] + [s for _, s in similar]


    vectors = np.array([model.wv[w] for w in words])


    if vectors.shape[1] > 3:
        pca = PCA(n_components=3)
        vectors_3d = pca.fit_transform(vectors)
        axis_titles = ("PC1", "PC2", "PC3")
```

```
    else:
        vectors_3d = vectors
        axis_titles = tuple(f"Dim{i+1}" for i in range(vectors.shape[1]))


    sizes = np.array(scores) * 25
    colors = np.array(scores)


    fig = go.Figure(
        data=[
            go.Scatter3d(
                x=vectors_3d[:, 0],
                y=vectors_3d[:, 1],
                z=vectors_3d[:, 2] if vectors_3d.shape[1] > 2 else np.zeros(len(words)),
                mode="markers+text",
                text=words,
                textposition="top center",
                marker=dict(
                    size=sizes,
                    color=colors,
                    colorscale="viridis",
                    showscale=True,
                    colorbar=dict(title="Similarity"),
                    line=dict(width=1, color="black"),
                    symbol=["diamond"] + ["circle"] * (len(words) - 1)
                ),
                hovertemplate="<b>%{text}</b><br>Similarity: %{marker.color:.2f}<extra></extra>",
            )
        ]
    )

    fig.update_layout(
        title=f"Similar Words to '{word}' (Top {top_n}) — Color/Size = Similarity",
        scene=dict(
            xaxis_title=axis_titles[0],
            yaxis_title=axis_titles[1],
            zaxis_title=axis_titles[2],
        ),
        height=600,
        margin=dict(l=0, r=0, t=40, b=0),
```

```
    )

    return fig



fig = explore_similar_words("love", model, top_n=30)
fig.show()
```

Beispielanalyse: Das Wort *love*

Diese Funktion zeigt, **wie nah verwandte Wörter im semantischen Raum** zueinander liegen. Hier das Beispielwort **"love"**.

Vergleich	Distanz	Warum
love ↔ loving	sehr gering	gleiche Wortfamilie, gleicher Kontext (Verbformen)
love ↔ baby	etwas größer	thematisch ähnlich, aber anderer Satzgebrauch

Semantische Zonen

Bereich	Wörter	Bedeutung
Zentrum	love, loving, lov, loves	direkter Wortstamm (höchste Ähnlichkeit)
Links-Vorne	baby, babe, darling, girl, heart	emotionale / beziehungsbezogene Substantive
Rechts-Oben	forever, never, always, ever	zeitliche oder abstrakte Begriffe
Mitte	want, feel, know, say, cause	häufige Verben im emotionalen Kontext
Unten	believe, true	Begriffe aus dem semantischen Feld „Vertrauen / Wahrheit“

Beispielinterpretationen:

- `love ↔ loving` → gleicher semantischer Kern, ähnliche Satzstruktur (Verbform)
- `love ↔ loves` → gleiche Bedeutung, anderer grammatikalischer Kontext

In [6]:

```python
def explore_embedding_space(model, n_words=30):
    vocab = list(model.wv.index_to_key)
    if len(vocab) == 0:
        raise ValueError("The model has an empty vocabulary.")
    n = min(n_words, len(vocab))
    words = vocab[:n]

    vectors = np.array([model.wv[w] for w in words])

    if vectors.shape[1] > 3:
        pca = PCA(n_components=3)
        vectors_3d = pca.fit_transform(vectors)
        axis_titles = ("PC1", "PC2", "PC3")
    else:
        vectors_3d = vectors
        axis_titles = tuple(f"Dim{i+1}" for i in range(vectors.shape[1]))

    distances = np.linalg.norm(vectors_3d, axis=1)

    fig = go.Figure(
        data=[
            go.Scatter3d(
                x=vectors_3d[:, 0],
                y=vectors_3d[:, 1] if vectors_3d.shape[1] > 1 else np.zeros_like(distances),
                z=vectors_3d[:, 2] if vectors_3d.shape[1] > 2 else np.zeros_like(distances),
                mode="markers+text",
                text=words,
                textposition="top center",
                marker=dict(
                    size=np.clip(distances * 3, 4, 24),
                    color=distances,
                    colorscale="Viridis",
                    showscale=True,
                    colorbar=dict(title="Distance"),
                ),
                hovertemplate="<b>%{text}</b><br>Distance: %{marker.color:.2f}<extra></extra>",
            )
        ]
    )
```

```
    fig.update_layout(
        title=f"Explore the Word Space (Top {n} words) — Size/Color = Distance from Origin",
        scene=dict(
            xaxis_title=axis_titles[0],
            yaxis_title=axis_titles[1] if len(axis_titles) > 1 else "",
            zaxis_title=axis_titles[2] if len(axis_titles) > 2 else "",
        ),
        height=600,
        margin=dict(l=0, r=0, t=40, b=0),
    )

    return fig

fig = explore_embedding_space(model, n_words=50)
fig.show()
```

3D-Visualisierung der Wortvektoren

Diese Funktion erzeugt eine **3D-Visualisierung** der Wortvektoren aus dem trainierten Word2Vec-Modell.

Da Word2Vec-Vektoren meist **50–300 Dimensionen** haben, wird eine **Hauptkomponentenanalyse (PCA)** durchgeführt, um die Daten auf **3 Dimensionen** zu reduzieren.

Funktionsweise

- Berechnet für jedes Wort den **Abstand vom Ursprung (0, 0, 0)** im 3D-Raum.
- Farbe und Größe der Punkte basieren auf diesem Abstand.
- Wörter mit ähnlicher Bedeutung oder gleichem Kontext liegen **nah beieinander**.

Aspekt	Bedeutung
Position	Semantische Lage im Raum — Wörter mit ähnlichem Kontext liegen nah beieinander
Abstand	Maß für semantische Ähnlichkeit — je näher, desto ähnlicher
Farbe	basiert auf Distanz vom Ursprung (Dunkelblau = nah, Gelb/Grün = weit entfernt)

Aspekt	Bedeutung
Größe	proportional zur Distanz — größere Punkte = auffälligere, semantisch ausgeprägte Wörter

Beispielhafte Cluster

- **Linke Seite:** „come“, „let“, „go“, „here“, „right“, „where“, „back“ → Funktions- oder Aktionsverben
- **Rechte Seite:** „bitch“, „fuck“, „shit“ → Vulgär- / Emotionalkontext
- **Zentrum:** „baby“, „love“, „wanna“, „think“ → emotionale, umgangssprachliche Wörter

Das Modell hat gelernt, dass bestimmte Wortgruppen ähnliche Verwendungen haben, selbst wenn ihre **Bedeutung unterschiedlich** ist.

In [7]:
```
tfidf_vect = TfidfVectorizer(
    tokenizer=lambda x: x,
    preprocessor=lambda x: x,
    token_pattern=None,
    lowercase=False
)

X_tfidf = tfidf_vect.fit_transform(df["tokens"])
terms = tfidf_vect.get_feature_names_out()

dim = model.wv.vector_size
doc_emb_tfidf = np.zeros((X_tfidf.shape[0], dim), dtype=np.float32)

for i in range(X_tfidf.shape[0]):
    row = X_tfidf[i]
    if row.nnz == 0:
        continue
    idxs = row.indices
    wts = row.data
    vecs = []
    w = []
    for j, wt in zip(idxs, wts):
        term = terms[j]
        if term in model.wv:
            vecs.append(model.wv[term])
            w.append(wt)
```

```
    if w:
        vecs = np.vstack(vecs)
        w = np.asarray(w, dtype=np.float32)
        doc_emb_tfidf[i] = (vecs * w[:, None]).sum(axis=0) / (w.sum() + 1e-9)

print("TF-IDF-Embeddings:", doc_emb_tfidf.shape)

# Optional: Nullvektoren rausfiltern (falls ein Song keine bekannten Wörter hat)
keep = np.linalg.norm(doc_emb_tfidf, axis=1) > 0
df_use = df.reset_index(drop=True).loc[keep].reset_index(drop=True)
emb_use = doc_emb_tfidf[keep]
print("Nach Filter:", emb_use.shape)
```

```
TF-IDF-Embeddings: (34049, 50)
Nach Filter: (34049, 50)
```

Dokument-Vektoren mit Word2Vec & TF-IDF

Ziel

Für jedes Dokument (z. B. Songtext) wird ein **repräsentativer Vektor** berechnet, der die Bedeutung aller Wörter kombiniert – **gewichtet nach ihrer Wichtigkeit**.

Einstellungen

Parameter	Bedeutung
`tokenizer=lambda x: x`	Verwendet die vorhandene Tokenliste (nicht erneut splitten)
`preprocessor=lambda x: x`	Kein Text-Cleaning oder Joinen
`token_pattern=None`	Deaktiviert die Standardtokenizer-Regel von sklearn
`lowercase=False`	Tokens bleiben in ihrer ursprünglichen Form (Groß-/Kleinschreibung bleibt erhalten)

Konzept

- **Word2Vec** → fängt **semantische Beziehungen** zwischen Wörtern ein

- **TF-IDF (Term Frequency – Inverse Document Frequency)** → hebt **wichtige Wörter** hervor

In [8]:
```python
def explore_doc_space(embeddings, labels=None, n_max=None):

    if embeddings is None or len(embeddings) == 0:
        raise ValueError("Keine Embeddings übergeben.")

    X = np.asarray(embeddings)
    if n_max is not None:
        X = X[:n_max]
        if labels is not None:
            labels = labels[:n_max]


    if X.shape[1] > 3:
        pca = PCA(n_components=3)
        X3 = pca.fit_transform(X)
        axis_titles = ("PC1", "PC2", "PC3")
    else:
        X3 = X
        axis_titles = tuple(f"Dim{i+1}" for i in range(X.shape[1]))
        # ggf. auf 3 Achsen auffüllen
        if X3.shape[1] == 1:
            X3 = np.hstack([X3, np.zeros((X3.shape[0], 2))])
        elif X3.shape[1] == 2:
            X3 = np.hstack([X3, np.zeros((X3.shape[0], 1))])


    distances = np.linalg.norm(X3, axis=1)


    if labels is None:
        labels = [f"Doc {i}" for i in range(X3.shape[0])]

    fig = go.Figure(
        data=[
            go.Scatter3d(
                x=X3[:, 0],
                y=X3[:, 1],
                z=X3[:, 2],
```

```
                mode="markers+text",
                text=labels,
                textposition="top center",
                marker=dict(
                    size=np.clip(distances * 3, 4, 24),
                    color=distances,
                    colorscale="Viridis",
                    showscale=True,
                    colorbar=dict(title="Distance"),
                    opacity=0.9,
                ),
                hovertemplate="<b>%{text}</b><extra></extra>",
            )
        ]
    )

    fig.update_layout(
        title="Explore the Document/Song Space — Size/Color = Distance from Origin",
        scene=dict(
            xaxis_title=axis_titles[0],
            yaxis_title=axis_titles[1] if len(axis_titles) > 1 else "",
            zaxis_title=axis_titles[2] if len(axis_titles) > 2 else "",
        ),
        height=650,
        margin=dict(l=0, r=0, t=40, b=0),
    )
    return fig

labels = None
for col in ["title", "tokens", "tag"]:
    if col in df_use.columns:
        labels = df_use[col].astype(str).tolist()
        break
if labels is None:
    labels = df_use.index.astype(str).tolist()

fig_docs = explore_doc_space(emb_use, labels=labels, n_max=25)
fig_docs.show()
```

explore_doc_space()

Die Funktion `explore_doc_space()` zeigt, **wie ähnlich oder unterschiedlich Songtexte zueinander sind**, basierend auf ihren **Embeddings**.

- Jeder Punkt = ein **Song (Dokument)**
- **Nähe zweier Punkte** → ähnliche Wortverwendungen oder Themen
- **Farbe & Größe** → Distanz vom Ursprung (semantische „Charakteristik“ des Songs)
- **Cluster** → Gruppen von Songs mit ähnlichem Inhalt, Stimmung oder Vokabular

Damit wird sichtbar, **welche Songs sich thematisch ähneln**,

> z. B. Liebeslieder, Partytracks oder melancholische Texte.