

LLM-gestützte Softwarearchitektur: Eine neue Ära? Oder: Brauchen wir noch Softwarearchitekten?

Von Jan Keim und Tobias Hey

Jahrestagung der GI-Fachgruppe "Architekturen" 2024



Textlastige Dokumente in Softwarearchitektur



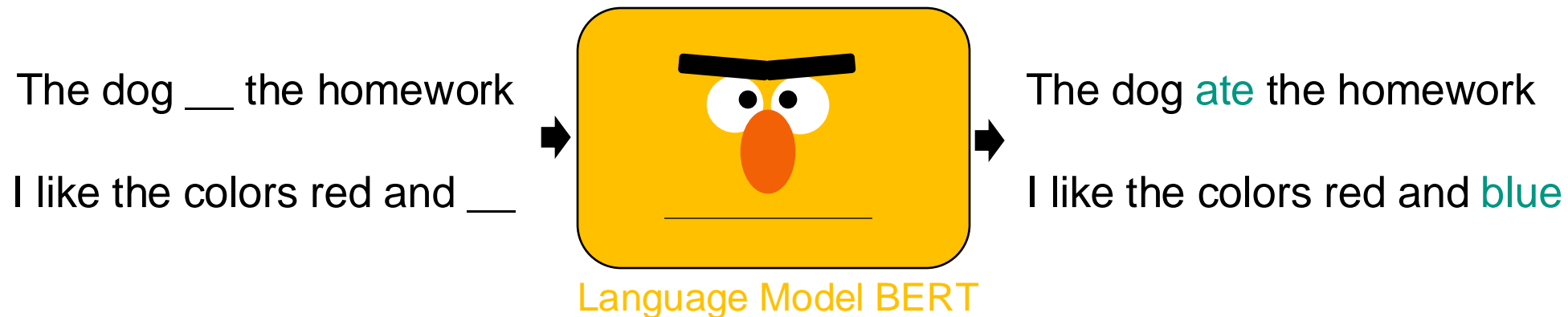
- Architekturbeschreibungen
- Dokumentierte Entwurfsentscheidungen
- Architekturwissen
- Architekturmodelle

➔ Potential für Sprachmodelle!



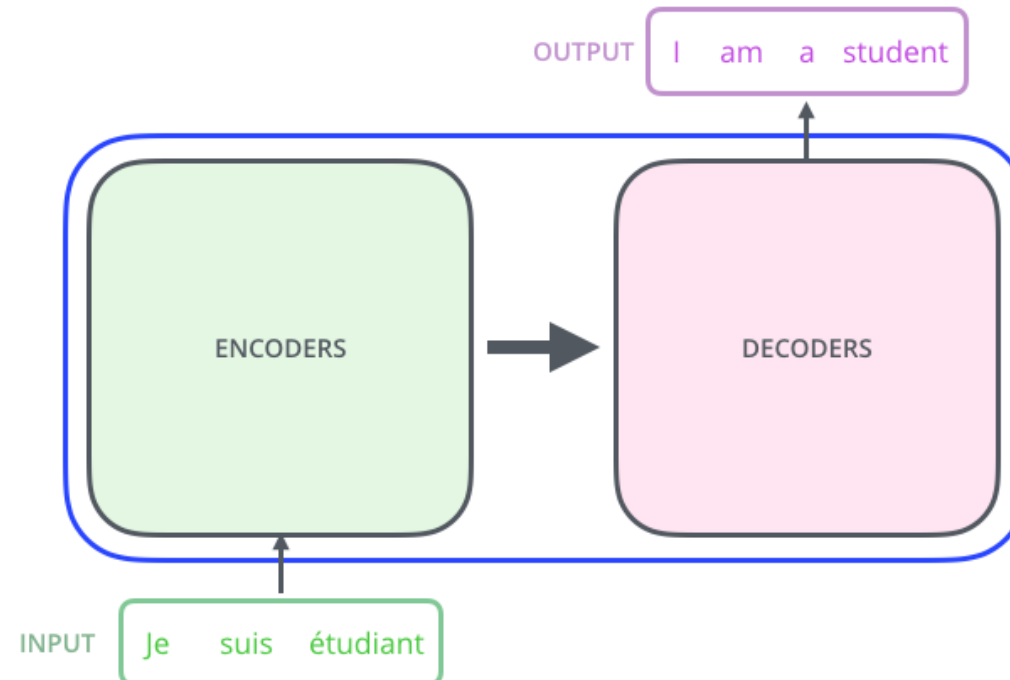
Was ist ein (Large) Language Model?

- Ein Language Model ist ein Ansatz, um Wörter/Sätze vorherzusagen
- Grundidee: „Lückentext lösen“ bzw. „Text weiterführen“



Transformer-Architektur: Was ist das?

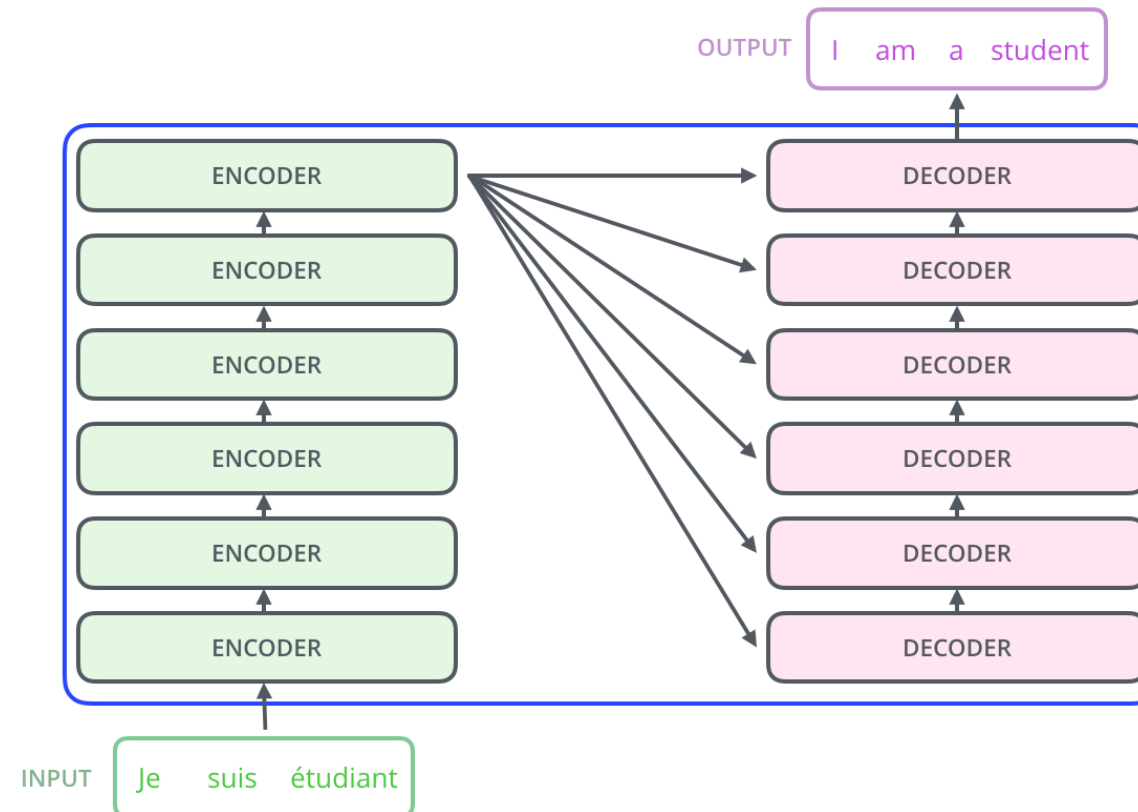
- Kernarchitektur der meisten LLMs wie BERT, GPT-3/4, Llama
- Eigentlich für Übersetzungen entwickelt
- Besteht aus De- und Enkodierungskomponente



Bildquellen: [jalammar.github.io](https://github.com/jalammar)

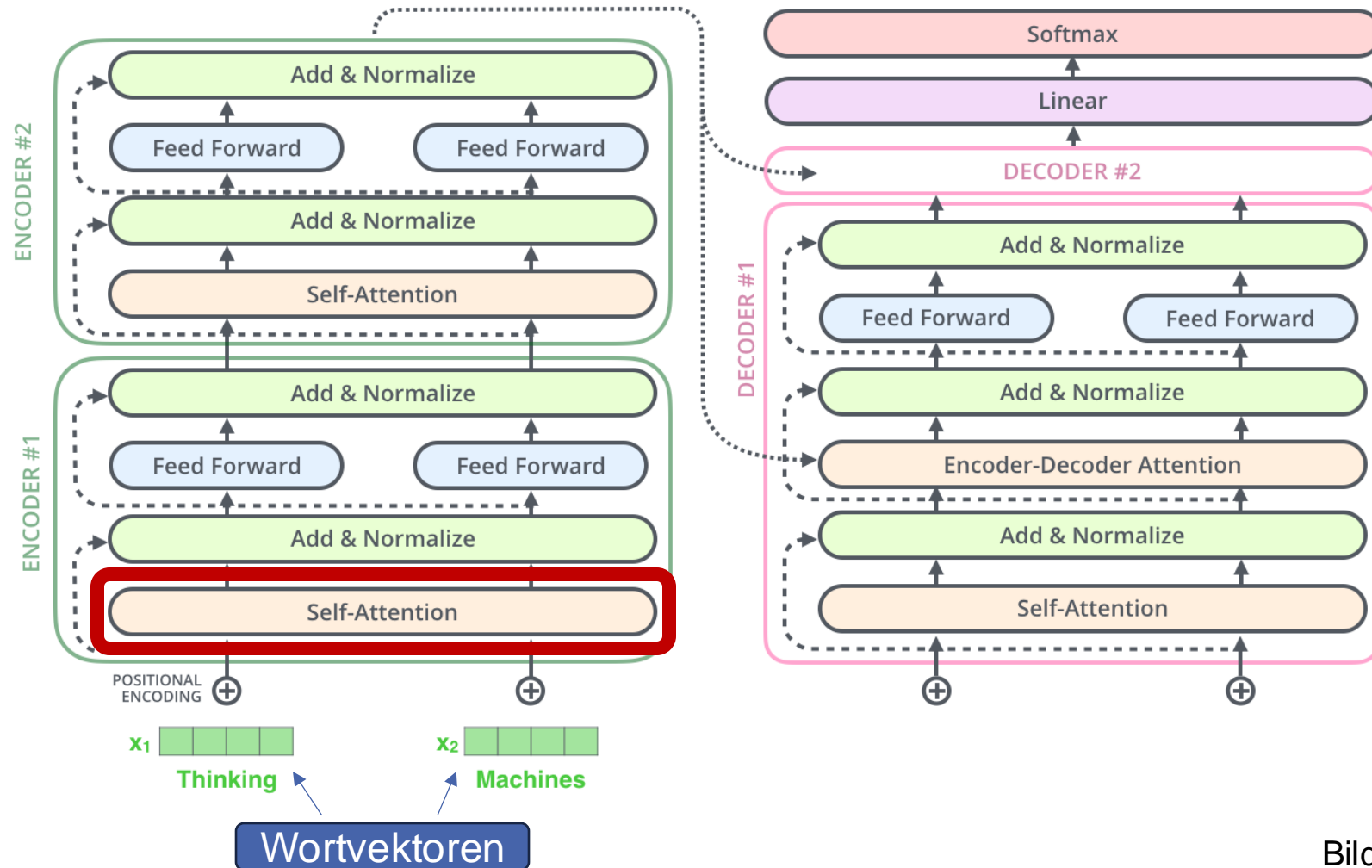
Transformer: De- und Enkodierungskomponente

- Komponenten sind ein Stapel an verbundenen Enkodierern/Dekodierern



Bildquellen: [jalammar.github.io](https://github.com/jalammar)

Transformer: Detailbild



Bildquellen: [jalammar.github.io](https://github.com/jalammar)

Wortvektoren bzw. Word Embeddings

- Modernes maschinelles Lernen benötigt Zahlen bzw. Zahlenvektoren
- Daher: Wörter als (*kurze*) Vektoren darstellen
- Berechnung mittels Maschinellernverfahren
- Wörter im Kontext *einbetten*

$$\begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 3 \\ \hline \end{array}
 -
 \begin{array}{|c|} \hline 0 \\ \hline 0 \\ \hline 1 \\ \hline \end{array}
 +
 \begin{array}{|c|} \hline 1 \\ \hline 0 \\ \hline 1 \\ \hline \end{array}
 =
 \begin{array}{|c|} \hline 2 \\ \hline 2 \\ \hline 3 \\ \hline \end{array}$$

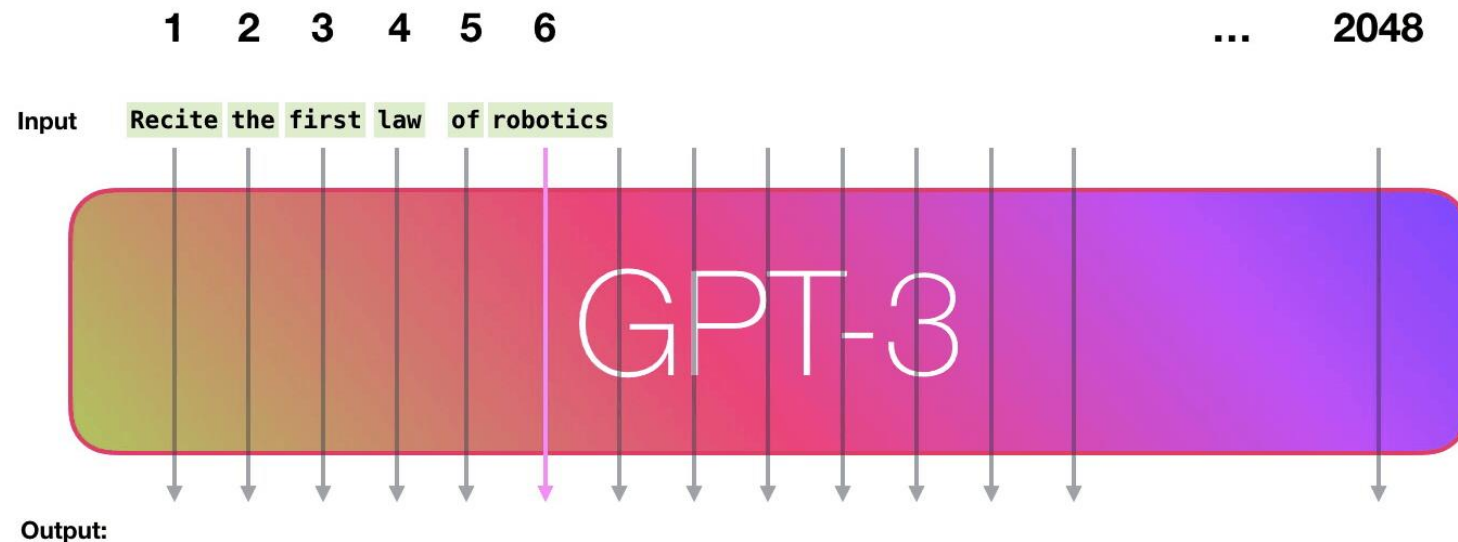
Berlin Deutschland Frankreich Paris

„In Berlin findet man den Regierungssitz der BRD.“

„Berlin wurde im 13. Jahrhundert erstmals urkundlich erwähnt.“

„In der Agglomeration Berlin leben rund 4,8 Millionen Menschen“

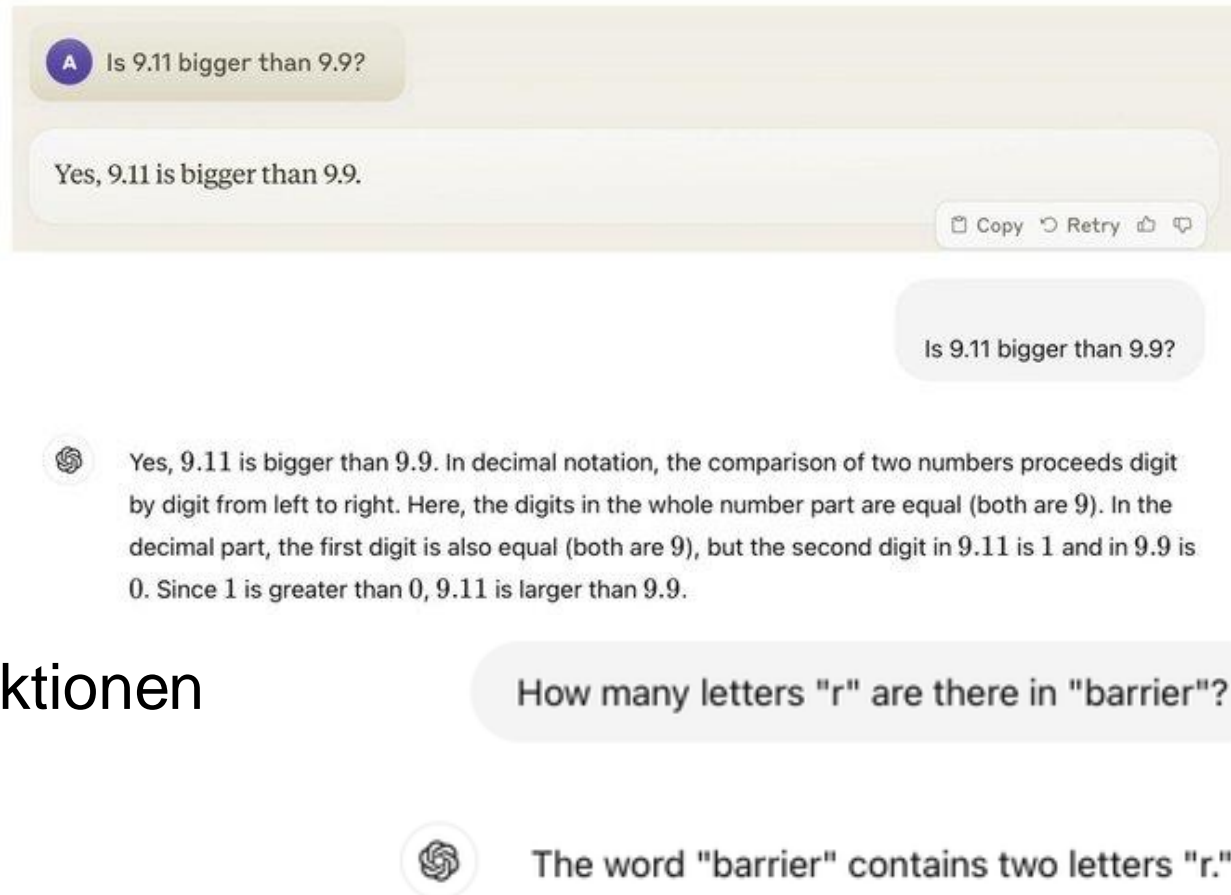
Wie produzieren Generative LLMs Text?



Images: [jalamar.github.io](https://github.com/jalammar)

Was können LLMs?

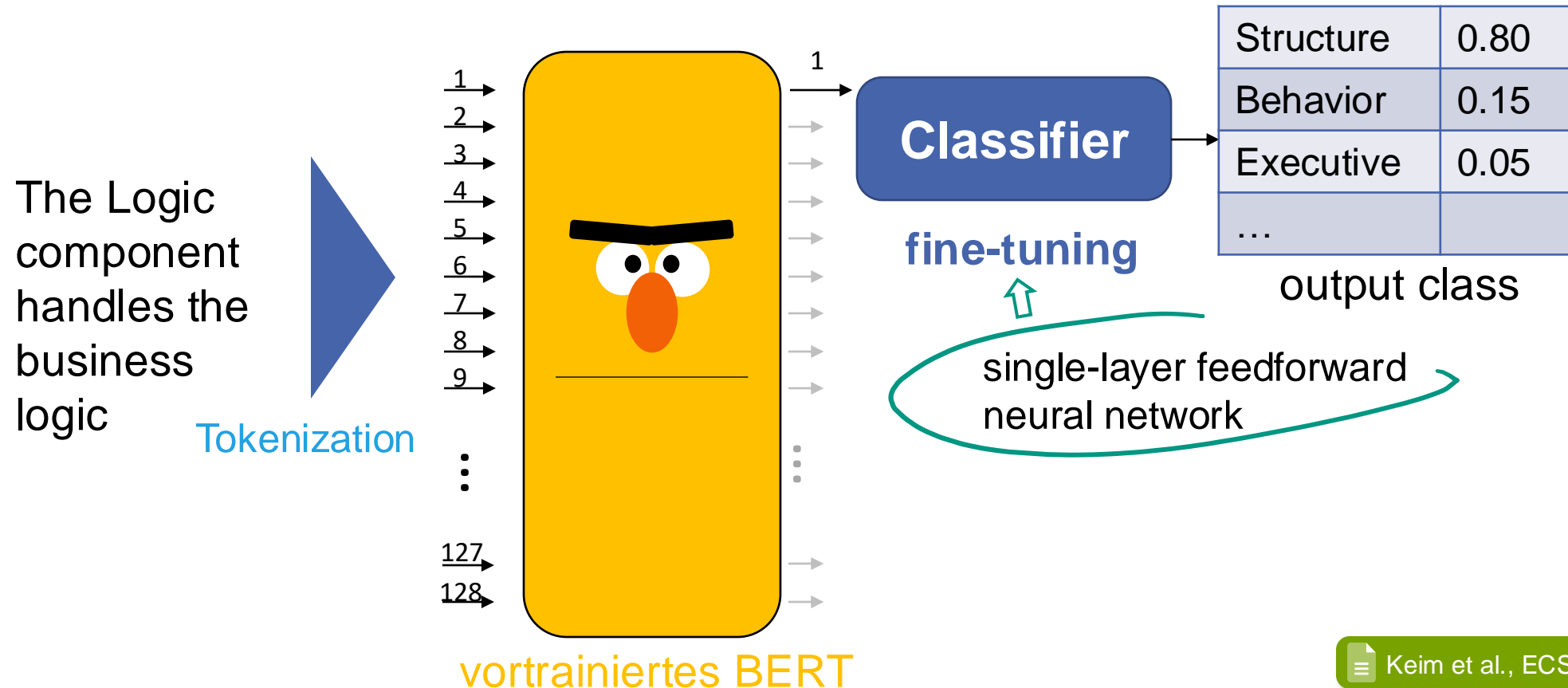
- Text generieren. Sonst nichts
- ABER: Die Sprache und der Trainingskorpus enkodieren Wissen
- LLM reflektiert nicht und kann eigentlich nicht entscheiden
- Erweiterung der Möglichkeiten
 - Function calling: Erkennen und Aufruf der benötigten (externen) Funktionen
 - Routing der Anfrage



Images: x.com/karpathy

Autom. Klassifizierung von Entscheidungen

- Problem: Wo befinden sich Entscheidungen in meiner Dokumentation?



Autom. Klassifizierung von Entscheidungen

	Lin. Regr. (trigram)			Decision Tree (BoW)			Rand. Forest (bigram)			BERT		
	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
Binary	0.877	0.894	0.885	0.850	0.852	0.851	0.831	0.970	0.895	0.901	0.942	0.921
Multi-class	0.452	0.451	0.427	0.314	0.322	0.304	0.346	0.353	0.269	0.575	0.559	0.552
Multi-label	0.578	0.326	0.396	0.453	0.394	0.406	0.482	0.090	0.145	0.679	0.427	0.500

- Binary: Entwurfsentscheidung oder nicht?
- Multi-Class: Klassifikation in eine von 24 Klassen der Taxonomie
- Multi-Label: Klassifikation in eine oder mehrere der 24 Klassen der Taxonomie

Identifikation von Architekturtaktiken

- Problem: Wo im Code sind Architekturtaktiken?
- Klassifizierung des Codes bezüglich Architekturtaktiken

	Prec.	Rec.	F1
AdaBoost	0.94	0.93	0.93
Bayesian Logistic Regr.	0.95	0.87	0.91
Tactic Detection	0.87	0.94	0.90
BERT	0.92	0.89	0.90

Traceability Link Recovery für Architektur

Software Architecture Documentation (SAD)

The **controller** receives incoming requests and verifies them.

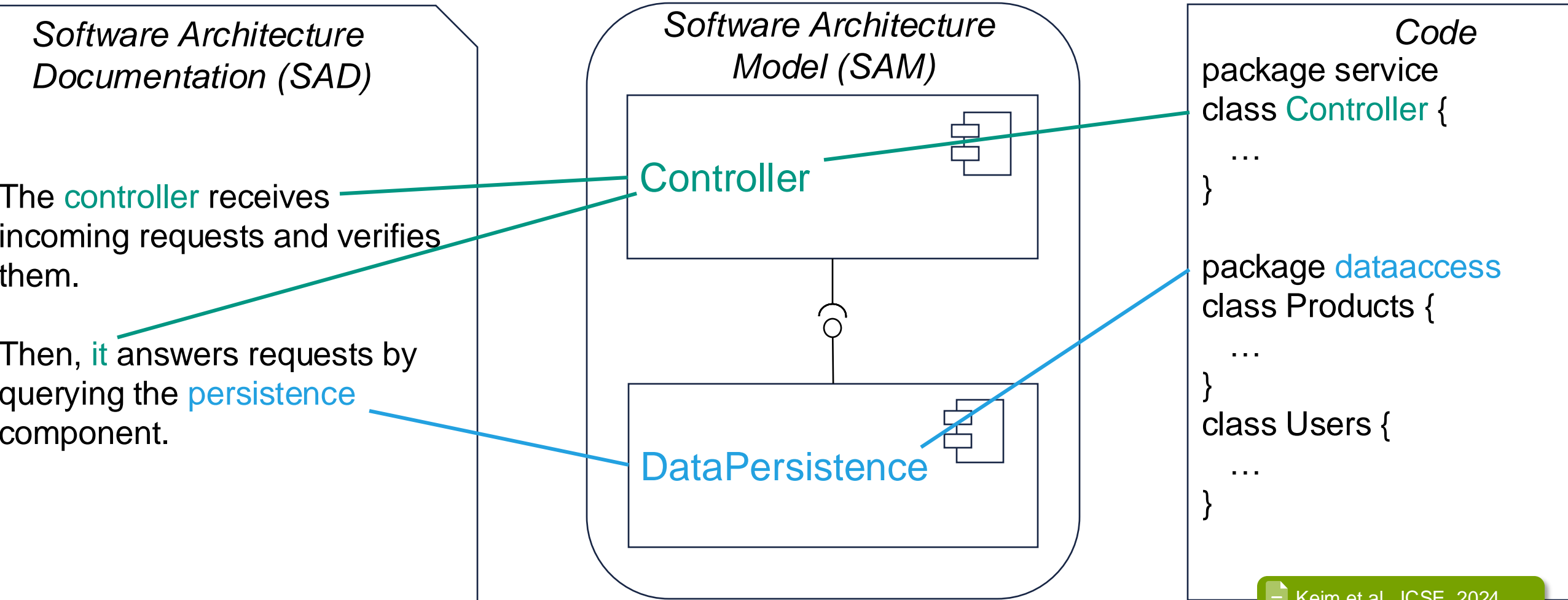
Then, **it** answers requests by querying the **persistence component**.

Code

```
package service  
class Controller {  
    ...  
}
```

```
package dataaccess  
class Products {  
    ...  
}  
class Users {  
    ...  
}
```

Traceability Link Recovery für Architektur



Keim et al., ICSE, 2024

Traceability Link Recovery

- Erste Idee: TLR als Klassifikationsproblem
- Problem: wenige bis keine Daten, komplexes Problem
- CodeBERT: LLM trainiert auf Code & Dokumentation
- Vergleichsansätze: ArDoCode & TransArC

	MediaStore			TeaStore			TEAMMATES			BigBlueButton			JabRef			Average			w. Average		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
CodeBERT	0.29	0.12	0.17	0.26	0.57	0.36	0.09	0.22	0.12	0.07	0.49	0.12	0.49	0.83	0.61	0.24	0.45	0.28	0.28	0.53	0.36
ArDoCode	0.05	0.66	0.09	0.20	0.74	0.31	0.37	0.92	0.53	0.07	0.57	0.13	0.66	1.00	0.80	0.27	0.78	0.37	0.47	0.92	0.62
TransArC	1.00	0.52	0.68	1.00	0.71	0.83	0.71	0.91	0.80	0.77	0.91	0.84	0.89	1.00	0.94	0.87	0.81	0.82	0.81	0.94	0.87

Keim et al., ICSE, 2024

TLR: Lösung Prompt Engineering?

- **Zero-Shot Prompting:** Dem LLM die Aufgabe direkt geben
- **One-/Few-Shot Prompting:** Dem LLM die Aufgabe + Beispiele geben

Sprachmodell: GPT-4	Prec.	Rec.	F1
Zero-shot	0.34	0.38	0.35
Few-shot	0.38	0.60	0.43
ArDoCo	0.83	0.82	0.80

TLR: Lösung Chain-of-thought Prompting?

- **Chain-of-Thought:** Modell nach einer Begründung/Herleitung fragen

Sprachmodell: GPT-4	Prec.	Rec.	F1
Zero-shot	0.34	0.38	0.35
Few-shot	0.38	0.60	0.43
Chain-of-Thought	0.56	0.56	0.53
ArDoCo	0.83	0.82	0.80

Retrieval-Augmented Generation (RAG)

- Problem: Information/Wissen ist nicht im LLM
→ Lösung: RAG
- Wissen wird vorverarbeitet in einer Datenbank gespeichert
- Bei Anfragen: Relevantes Wissen wird aus der Datenbank gesucht (Retrieval) und der Anfrage hinzugefügt
- LLM kann so dieses Wissen miteinbeziehen
→ Vorteil: Einschränkung der Antwort auf gegebene Quellen

Anwendung TLR-RAG - Ergebnisse

Sprachmodell: GPT-4	Prec.	Rec.	F1
Zero-shot	0.34	0.38	0.35
Few-shot	0.38	0.60	0.43
Chain-of-Thought	0.56	0.56	0.53
RAG + Zero-shot	0.44	0.85	0.55
RAG + Few-shot	0.34	0.90	0.47
RAG + CoT	0.52	0.90	0.64
ArDoCo	0.83	0.82	0.80


☰ Preliminary

„Intelligente“ Suchmaschine

- Vorbereitung: Wissensbasis wird erstellt
- Nutzer stellt Frage an das System
 - Identifizierung von relevantem Wissen aus der Wissensbasis
 - Wissen wird dem LLM zur Verfügung gestellt
 - LLM generiert aus Frage + Wissen eine Antwort

Ask your doc(umentation)

Ask your code and documentation anything!

 What architectural patterns exist in the system?

Architectural Patterns in the System

The CWA Server system follows several architectural patterns to ensure robustness, scalability, and maintainability. Some of the key architectural patterns include:

1. **Microservices Architecture:** The system is designed as a set of cron jobs and services exposing APIs deployed within a Kubernetes cluster. This enables modularity, independent deployment, and scalability of different components.
2. **Exposure Notification Framework:** The main objective of the system is to enable users to take part in the exposure notification framework based on Apple/Google specifications. This involves compliance with specific protocols and data privacy concerns.
3. **Integration with Other Systems:** The system integrates with external systems like Object Stores, Federation Gateways, and Verification Servers to enable interoperability and data sharing across different EU nations.
4. **Client-Server Architecture:** The system consists of client-side components (mobile apps) that interact with server-side components (CWA Server) to submit keys, aggregate data, and distribute configuration files.
5. **Event-Driven Architecture:** The system handles events such as key submissions, aggregation, and distribution in an event-driven manner to ensure timely processing and response to changing data.
6. **Security Architecture:** The system incorporates security-related architectural patterns such as TLS configuration, endpoint protection, authenticity checks, and fake submission detection to ensure data integrity and user privacy.

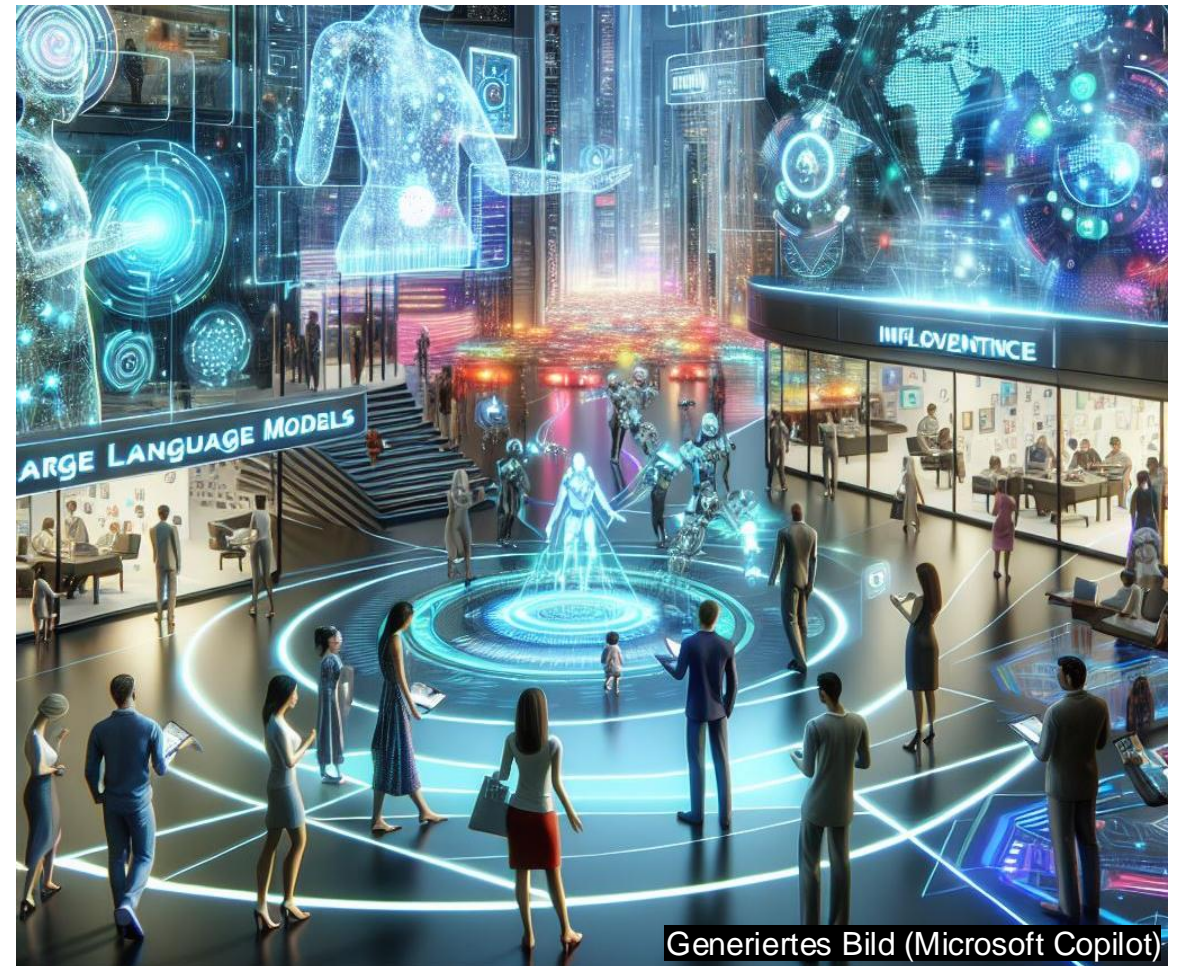
These architectural patterns work together to form a robust and efficient system for handling exposure notifications and data sharing within the CWA Server ecosystem.

Sources:

- [1] data\cwa-server\CODE_OF_CONDUCT.md
- [2] data\cwa-server\CONTRIBUTING.md
- [3] data\cwa-server\docs\ARCHITECTURE.md
- [4] data\cwa-server\docs\README.md

Eine neue Ära?

- Ja
 - Verändert vieles, disruptive Technologie
 - Ermöglicht neue Ansätze und Verbesserung best. Ansätze
 - Vorteil: Vieles wird “einfacher“, leichter Einstieg



Eine neue Ära?

Aber: Herausforderungen, Probleme

- Halluzinationen und andere Ungenauigkeiten
- Begrenzte Kontextlänge
- Urheberrecht und geistiges Eigentum
- Kosten
- Open Models (Llama) vs. Closed Models (GPT)
- Keine neuen Ideen, Entscheidungsfindung, Reflektion,..
- Prototypischer Erfolg vs. Reale Anwendung

→ Wir brauchen gut ausgebildete SoftwarearchitektInnen

→ LLMs sind einfach zu verwenden, aber das bedeutet nicht, dass sie immer die beste Lösung für ein bestimmtes Problem sind





Folien, Links, etc.



ardoco.de/c/fg-arch24

One more thing:
geplanter Arbeitskreis

Data Science for SWA