Hebrew SMS Spam Filter

https://github.com/amitbashan/sms-hebrew-spam-filter

Authors: Amit Bashan, Hila Damin, Tomer Sasson
Advisor: Dr. Gershon Kagan

Table of Contents

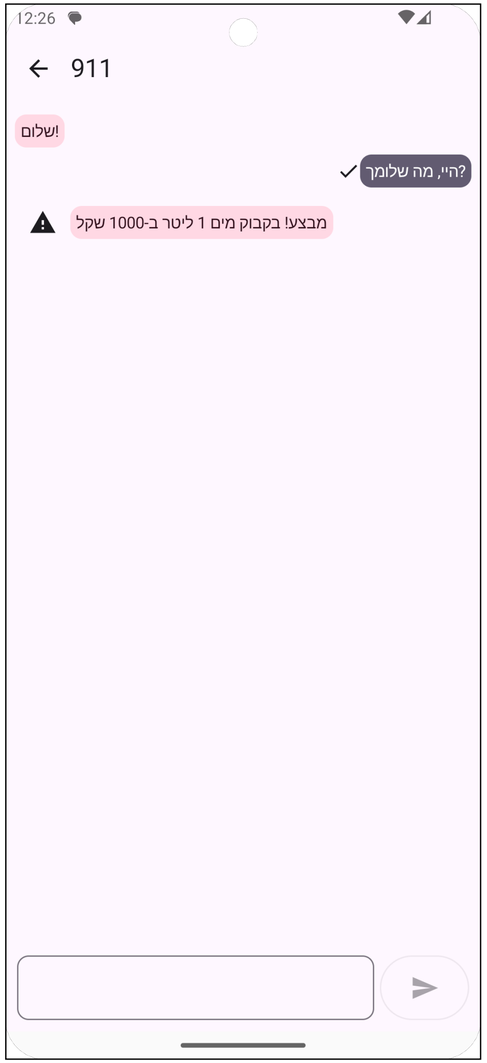
Problem Definition

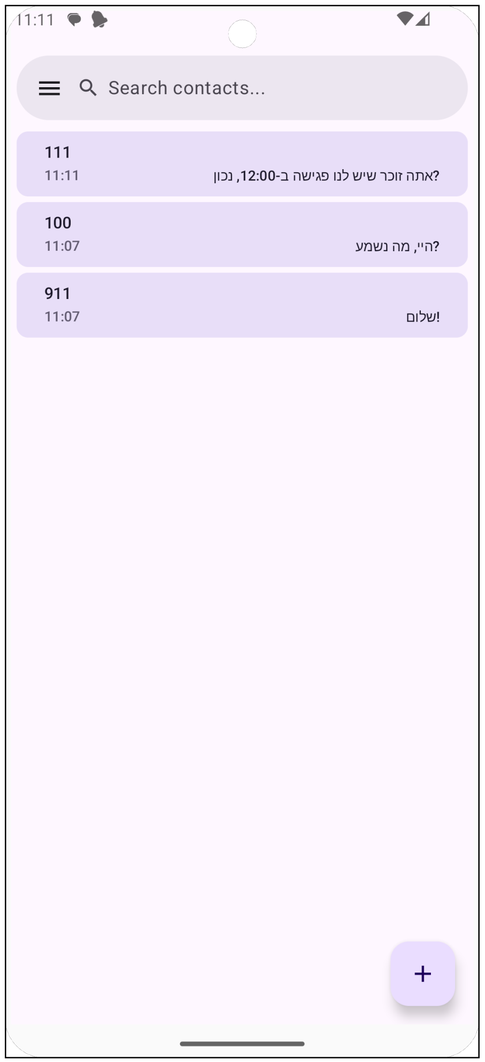
- We wish to detect Hebrew spam messages sent via SMS
- SMS messages are most commonly exchanged on phones
- Taking inspiration from the default SMS apps on phones, we decide to:
  - Choose Android as our mobile platform, as default apps can be changed
  - Develop a standard SMS app, to receive/send messages
  - Integrate a machine learning model trained to detect Hebrew spam
  - Use the model to detect & block spammers

Developing an SMS App

Designing the app

- We design a standard main screen layout with a contact list, search feature, menu bar, and an option to add a contact
- The chat screen features one-on-one conversation with messages from the contact on the left and ours on the right, plus indicators for spam and message delivery status

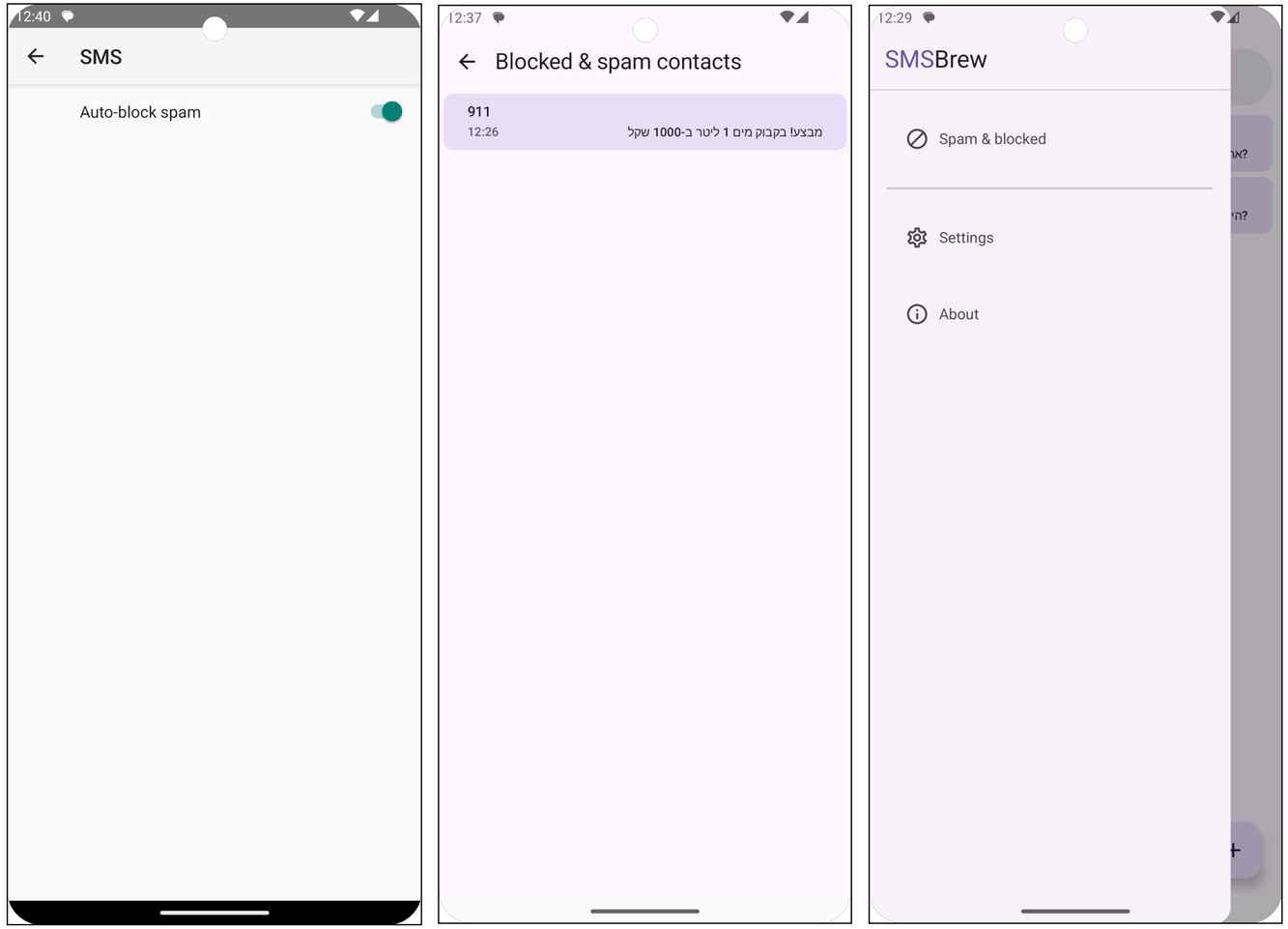




Developing an SMS App

Designing the app

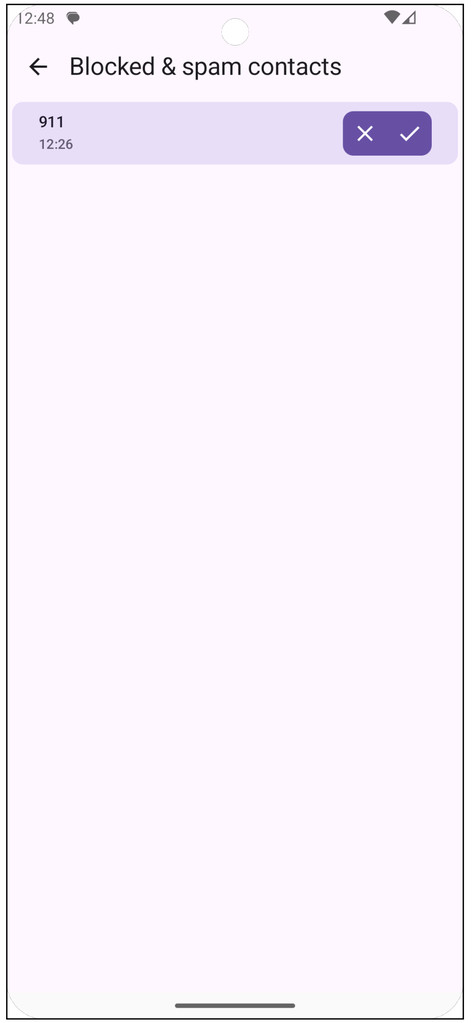
- The menu includes spam & blocked contacts, settings, and about screens
- In settings, there's an option to automatically block contacts that send messages classified as spam

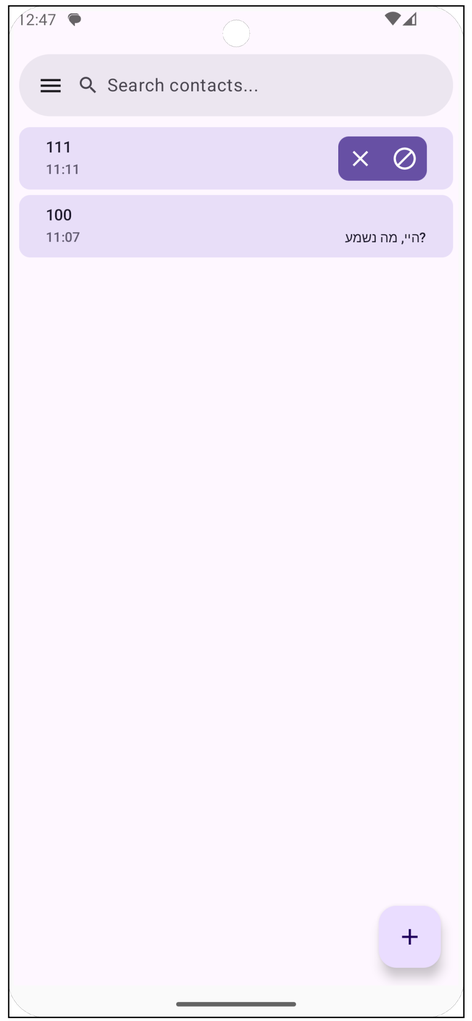


Developing an SMS App

Designing the app

- A long press on a contact button triggers a menu to block or unblock the contact
- Notifications will not appear if the contact is blocked

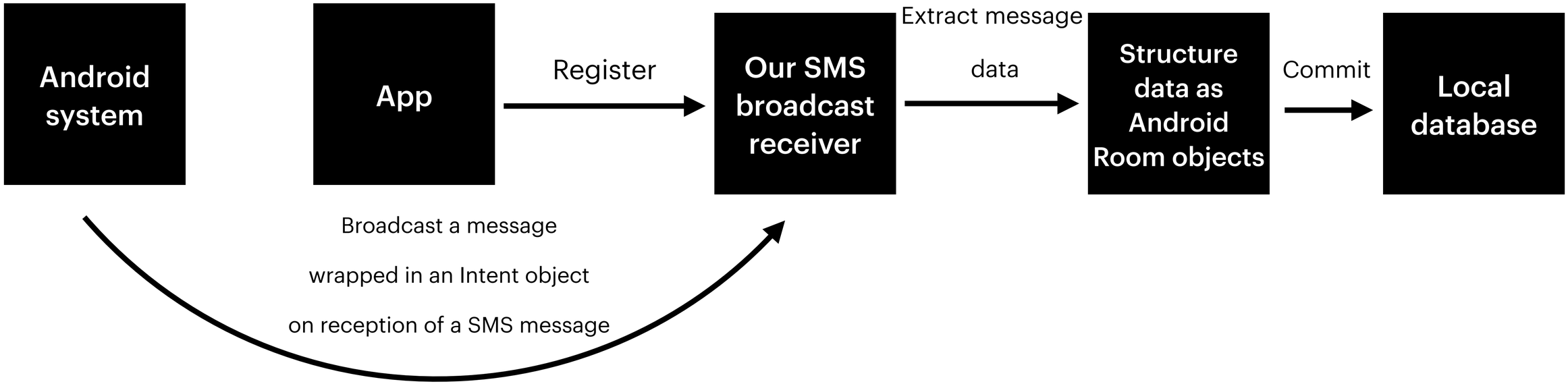




Developing an SMS App

Receiving SMS messages

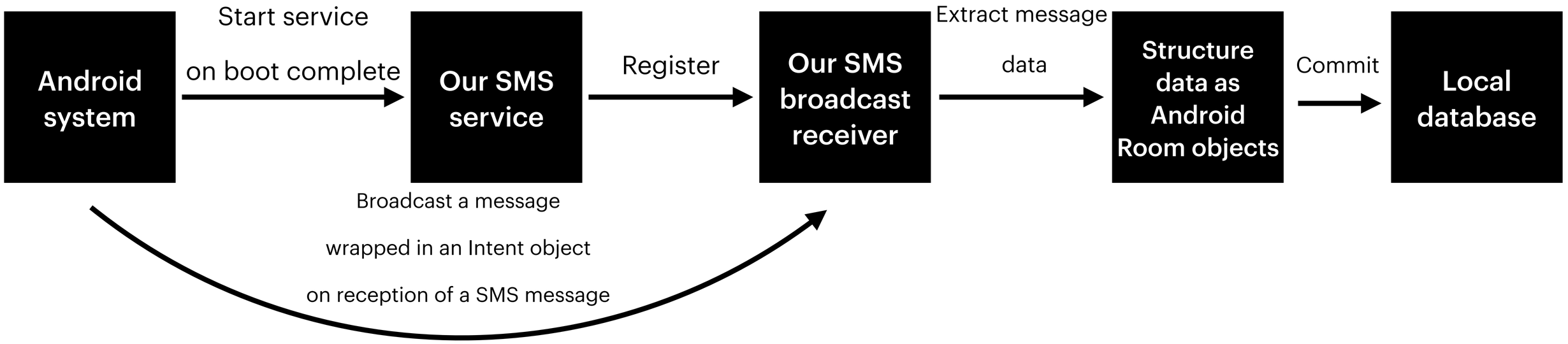
- In Android, apps receive messages from other apps or the system itself via broadcasts
- The broadcast message itself is wrapped in an `Intent` object whose action string identifies the event that occurred
- In our case, the action string is `SMS_RECEIVED_ACTION`



Developing an SMS App

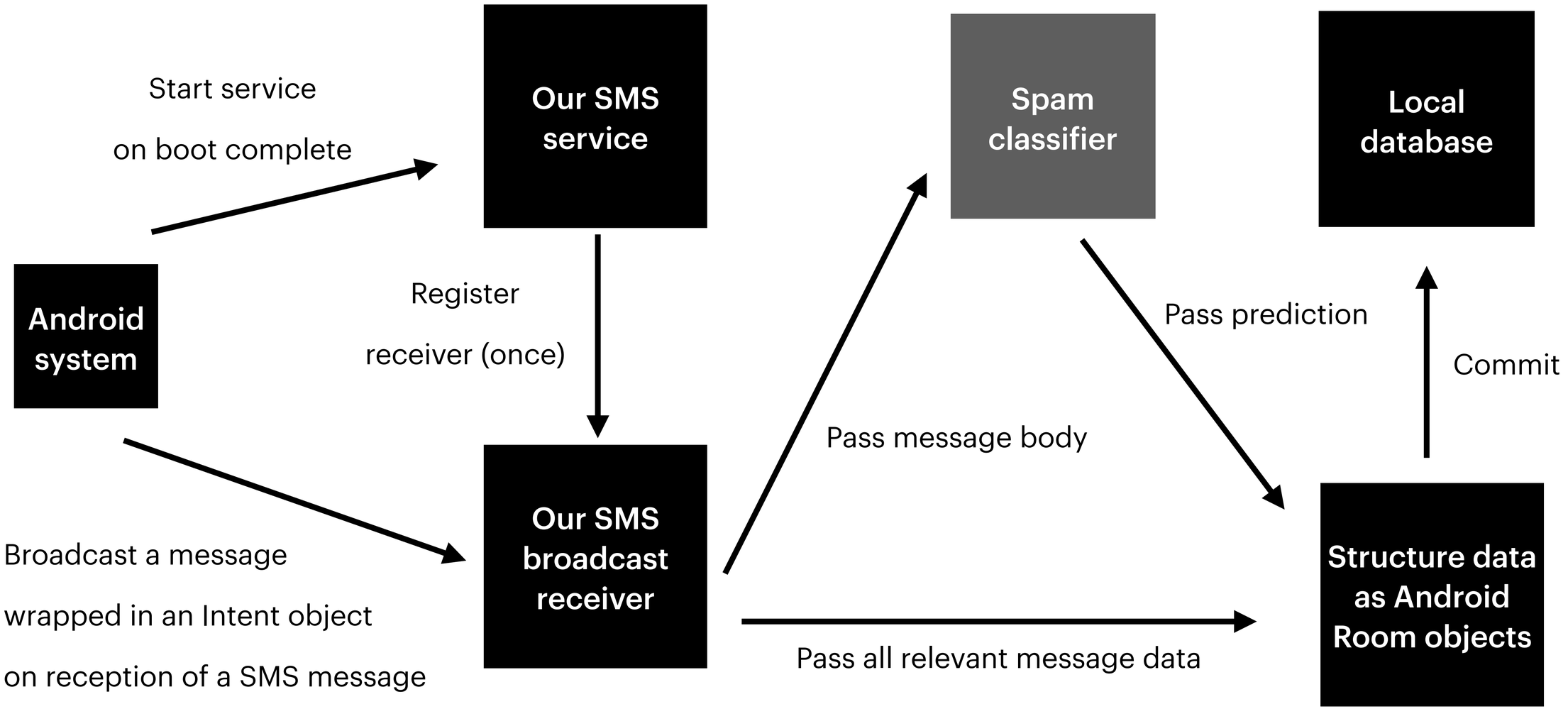
Receiving SMS messages - when the app is not running

- We've registered our custom broadcast receiver and are receiving SMS messages, but the app doesn't receive SMS messages when closed
- To solve this, we need to add a background service to listen for broadcasts continuously



Developing an SMS App

Receiving SMS messages



Developing an SMS App

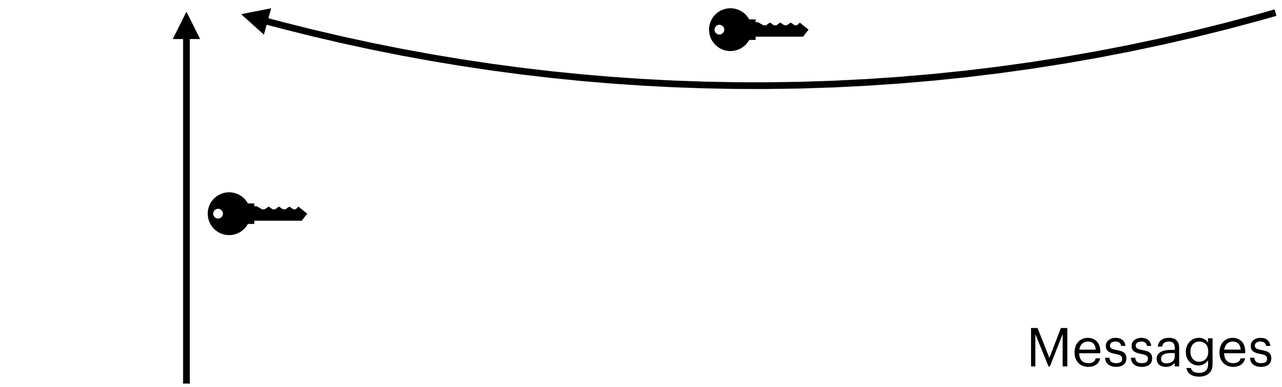
Database schema

Contacts

Originating address	Is blocked?

Contact Previews

Originating address	Timestamp	Message

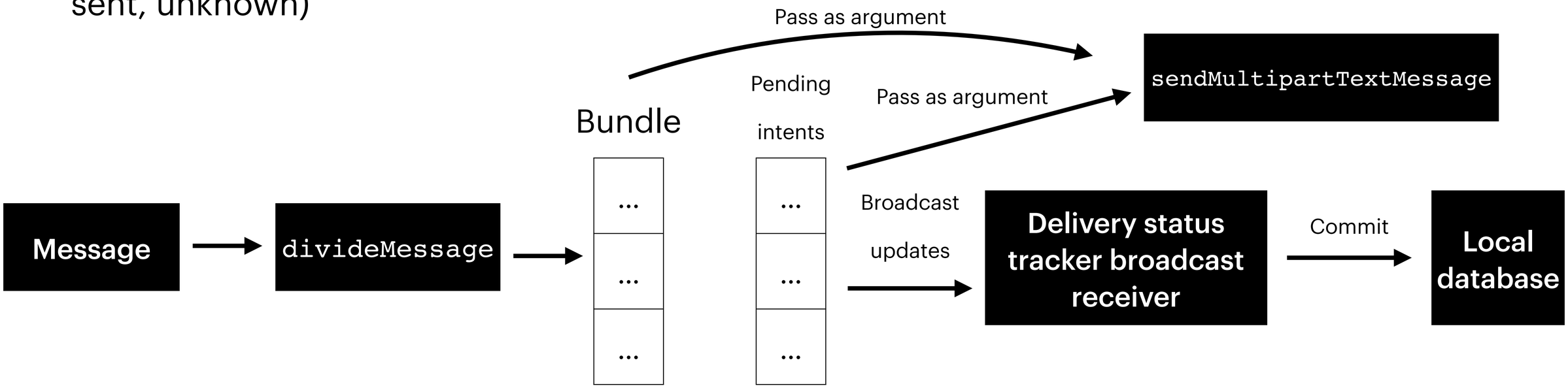


Messages

Originating address	**Timestamp**	Message	Is me?	Message status (delivered, sent, unknown)	Is spam?	Probability of being spam

Developing an SMS App
Sending SMS messages

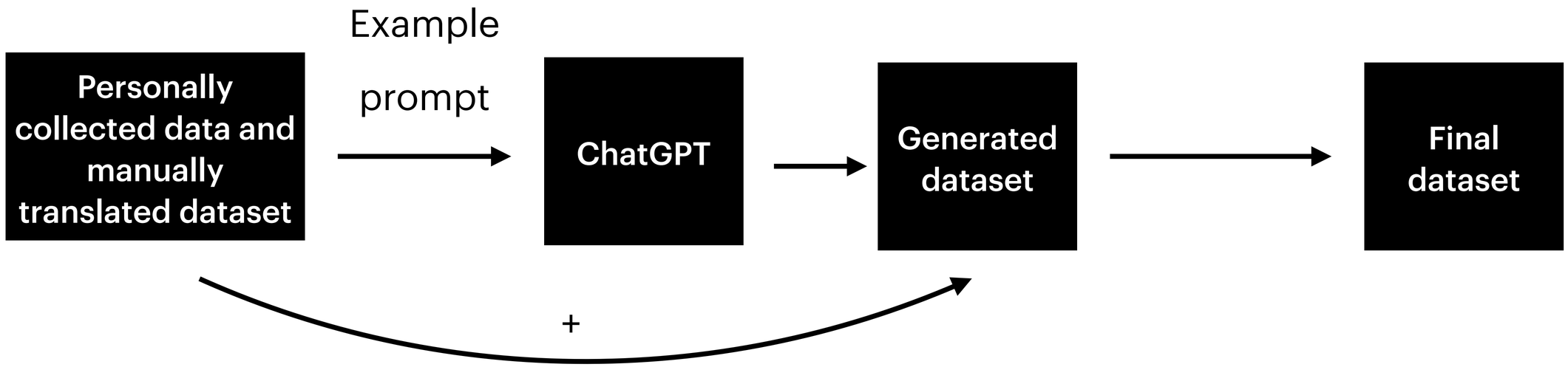
- To send SMS messages on Android, we use the `SmsManager` class
- We use `sendMultipartTextMessage` to send messages, including those over the 160 character limit by splitting them with `divideMessage`
- We pass pending intents for each message part to track their delivery statuses (delivered, sent, unknown)



Integrating a machine-learning model

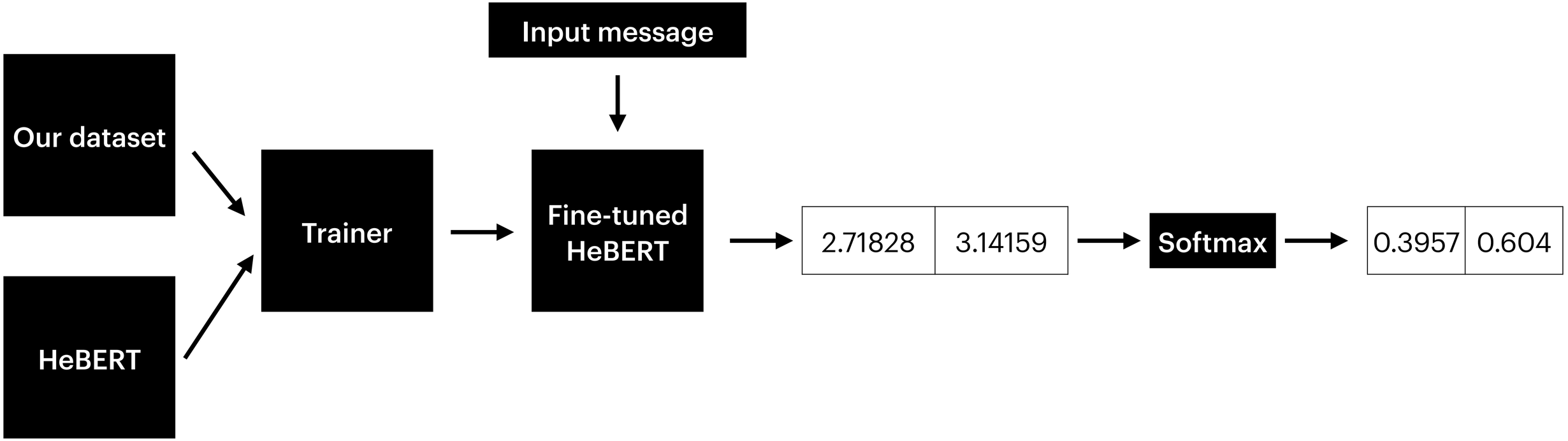
Gathering data

- Unfortunately, we found no publicly available Hebrew SMS spam datasets
- Consequently, we collected and labeled SMS messages from our personal phones
- This was insufficient for training, so we used ChatGPT to generate additional data and manually translated an English SMS spam dataset



Integrating a machine-learning model

- We use HeBERT, a pre-trained transformer model, to predict if a message is spam
- We fine-tune the model for our classification task to produce a vector that, after applying softmax, represents probabilities



Integrating a machine-learning model

Fine-tuning

- We utilize the HuggingFace Transformers library with PyTorch to:
  - Download the model
  - Employ the corresponding tokenizer with its configuration
  - Train the model
- Lastly, we use the ONNX library to export and quantize our trained model
- Quantization is the process of converting the model's weights and activations from a higher precision to a lower precision to lower memory and storage usage, and decrease inference time
- ONNX is an open source format for AI models, we can use it to perform inference with Kotlin using the KotlinDL library in our app

Integrating a machine-learning model

Evaluating and validating our model

10% of the dataset is for testing

90% of the dataset is for training

Epochs = 5

Learning rate $= 5 \cdot 10^{-5}$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{123}{123 + 3}$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{123}{123 + 12}$$

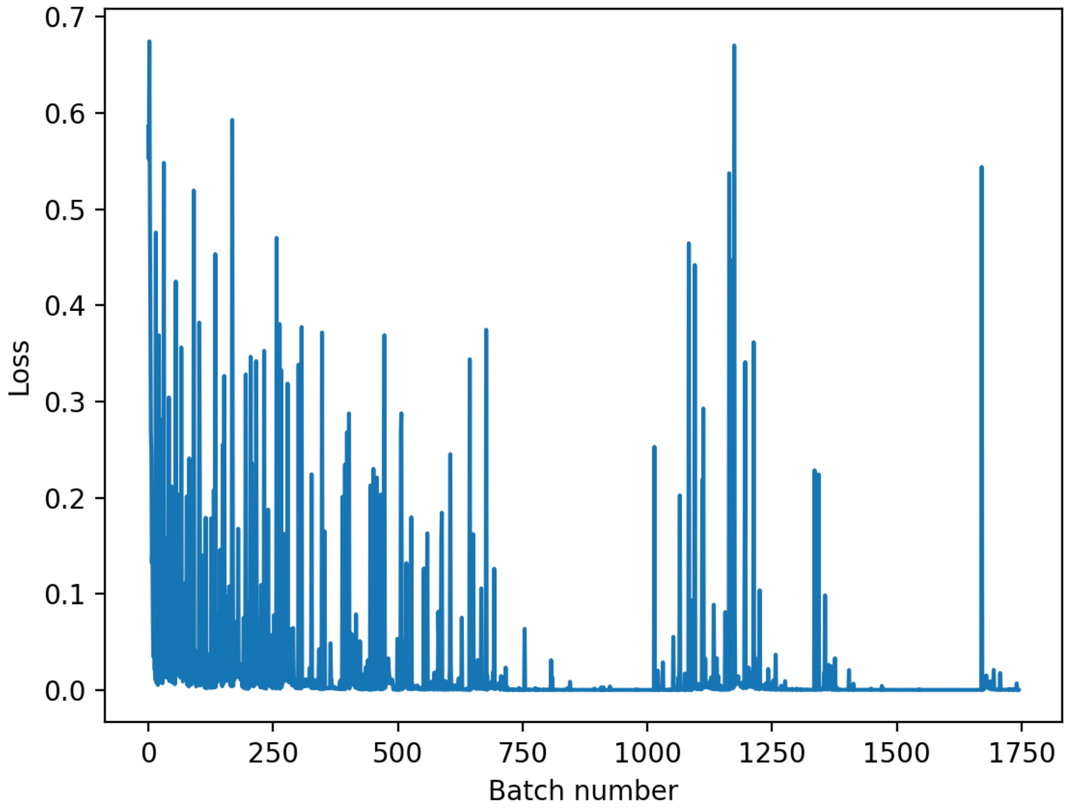
$$\text{F1 Score} = 2 \circ \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \approx 0.9405$$

Confusion matrix

True label \ Predicted label	0	1
0	483	3
1	12	123

Loss as a function of

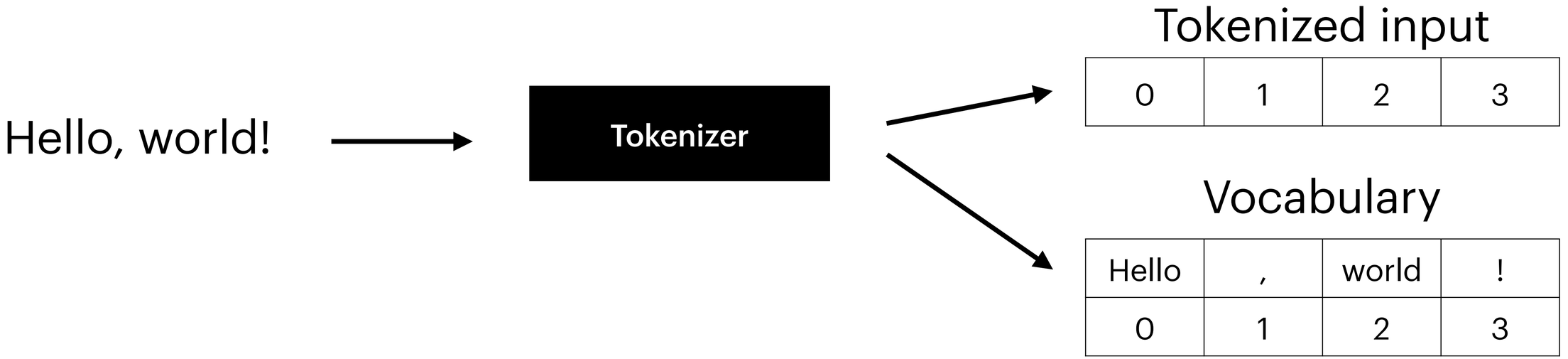
the batch number (batch size = 16)



The Architecture of a Transformer

Text preprocessing

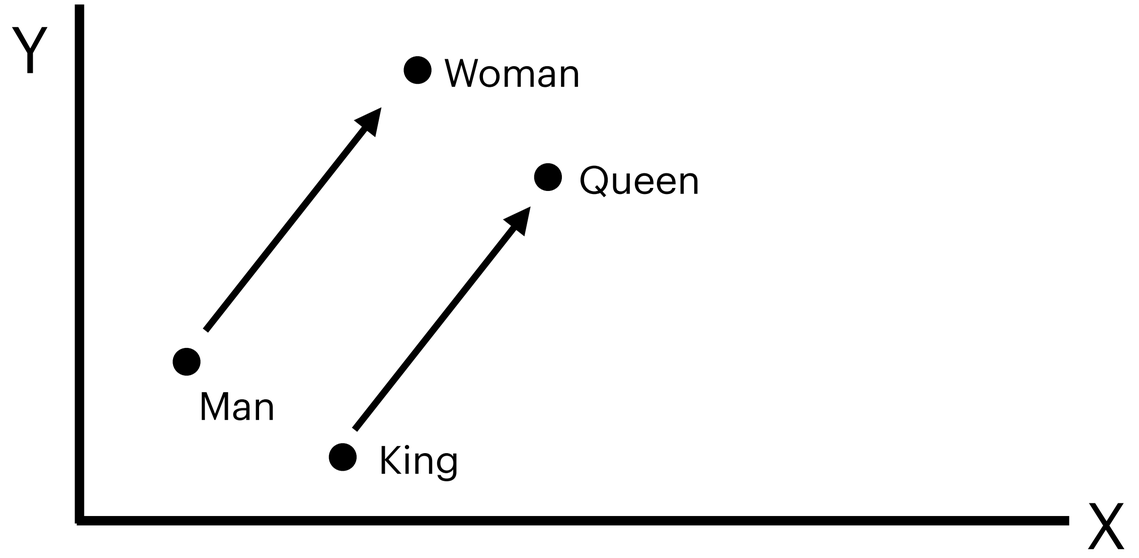
- The first step in NLP is preprocessing your textual data. Often, the preprocessing is **tokenization**
- Tokenization is the process of breaking down text into smaller chunks such as words
- By tokenizing our raw data, we split it into meaningful and minimal units that a machine can understand and process, and we represent each token with a unique number



The Architecture of a Transformer

Text preprocessing

- The second step is **vectorization**, giving each token an **embedding**
- An embedding is a vector of real numbers whose dimension reflects the number of features a token can exhibit in the model
- During the training process, the embeddings of each token acquire semantic meaning as tokens with similar characteristics are positioned closer together in the embedding space



The Architecture of a Transformer

Text preprocessing

- The third and last step is **positional encoding**
- Positional encoding injects information into the embeddings of tokens so the model can understand how its input is ordered
- Transformer models process all tokens simultaneously, so they need a way to keep track of the position of each token
- The positional encoding is calculated by a function which produces a unique vector for each position, typically using sinusoidal functions as introduced by the paper "**Attention Is All You Need**"

Token	Index
Hello	0
,	1
world	2
!	3

Positional Encoding →

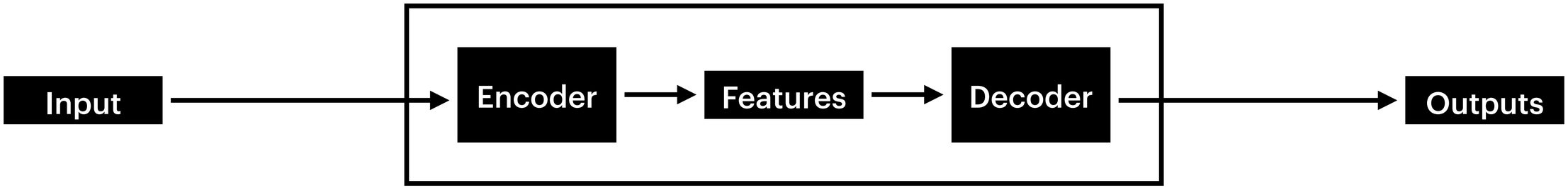
0	1	0	1
1	0.54	0.10	1
0.91	-0.42	0.20	0.98
0.14	-0.99	0.30	0.96

\+ Token embeddings = Position encoded embeddings

The Architecture of a Transformer

The encoder and decoder

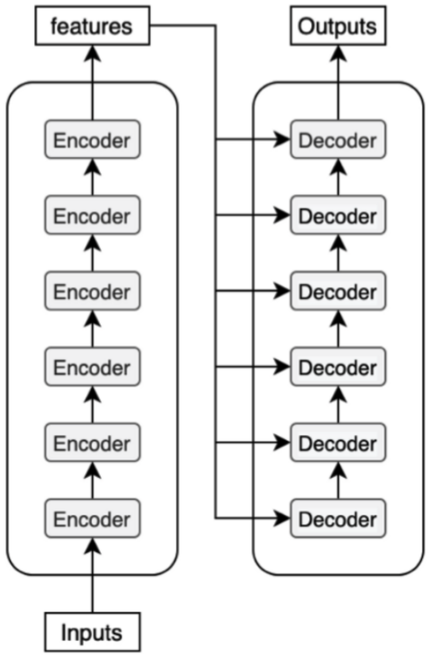
- The transformer is an encoder/decoder network at a high-level
- The encoder extracts features from an input sentence, and the decoder uses the features to produce an output sentence (or a probability distribution in a classification task, etc.)



The Architecture of a Transformer

The encoder and decoder

- The encoder in the transformer consists of multiple encoder blocks
- An input goes through the encoder blocks, and the output of the last encoder block becomes the input features to the decoder
- The decoder also consists of multiple decoder blocks, each decoder block receives the features from the encoder



The Architecture of a Transformer

The internals of the encoder and the decoder

- The multi-head self attention component is present in both the encoder and decoder
- This component is responsible for capturing relationships between tokens in a sequence. For example, it can identify that “bank” in “river bank” has a different meaning than in “financial bank”
- The masked multi-head attention component is only present in the decoder
- It is crucial because it ensures that the decoder generates text in a logical, sequential manner. Without it, the model could reference future words it shouldn’t have access to, leading to incoherent and confusing outputs

The Architecture of a Transformer

The steps after decoding

- The decoder outputs a vector that contains information that is helpful in generating the next token, it contains contextual information, from the input and what it has already generated
- Secondly, that output vector is transformed by a linear layer to match the size of the vocabulary, instead of having the size of the dimension of the embedding space
- Lastly, the softmax layer further converts the vector into a probability distribution
- Next, typically the word in the vocabulary which has the highest probability in the probability distribution vector will be chosen and the decoding process is reiterated with the newly chosen word/token

Video Demonstration