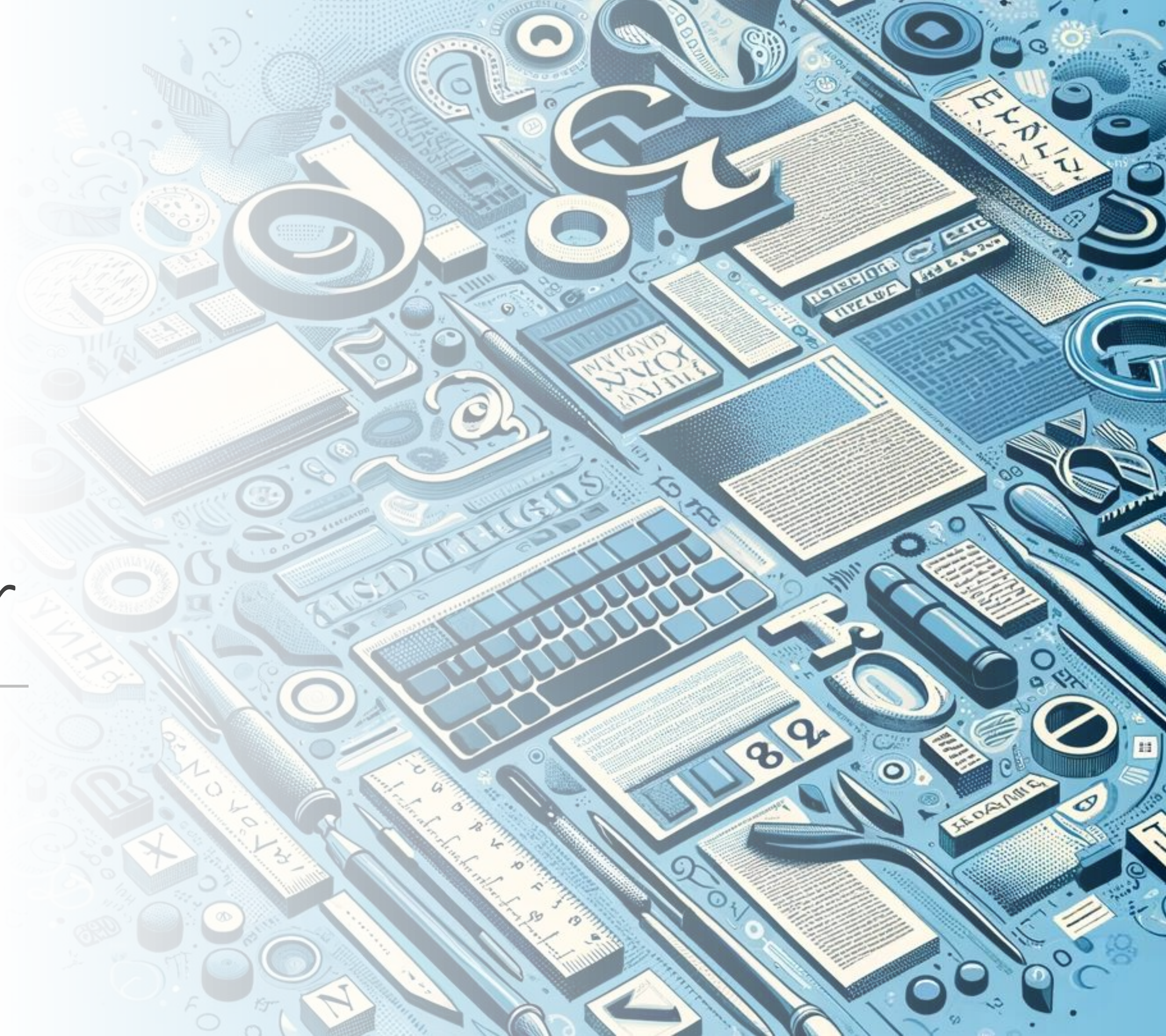
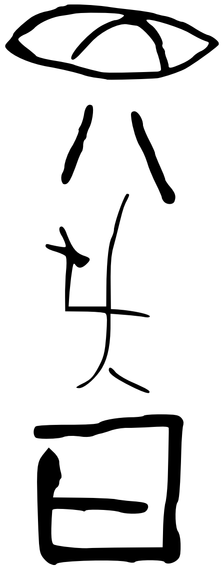
DATA SCIENCE RETREAT®
SINCE 2014

AI Font Generator

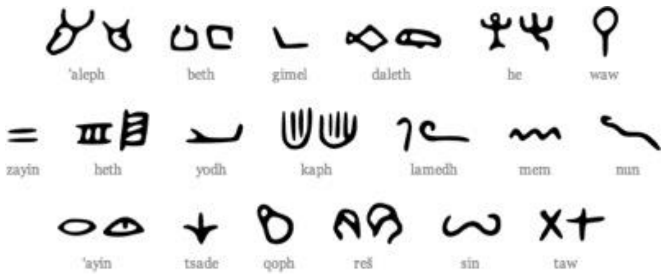


Brief history

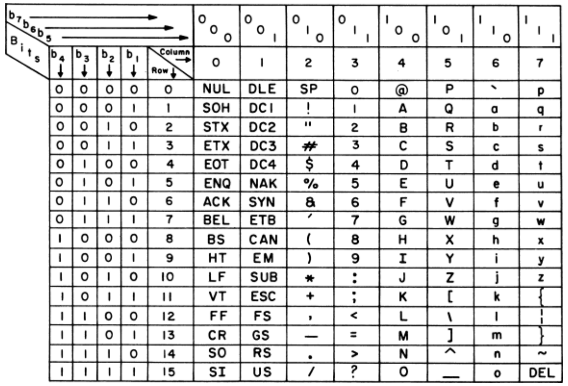


Jiahu symbols[1]

ca. 9000 BC:	Token system for accounting (Mesopotamia)
ca. 6600 BC:	Jiahu Symbols (China). First writing system
ca. 2000 BC:	Proto-Sinaitic script (Egypt): First alphabet
ca. 700 BC:	Latin Alphabet (Roman Empire)
1963	First edition of ASCII standard
(1990	Wingdings ☺)



Proto-Sinaitic[2]

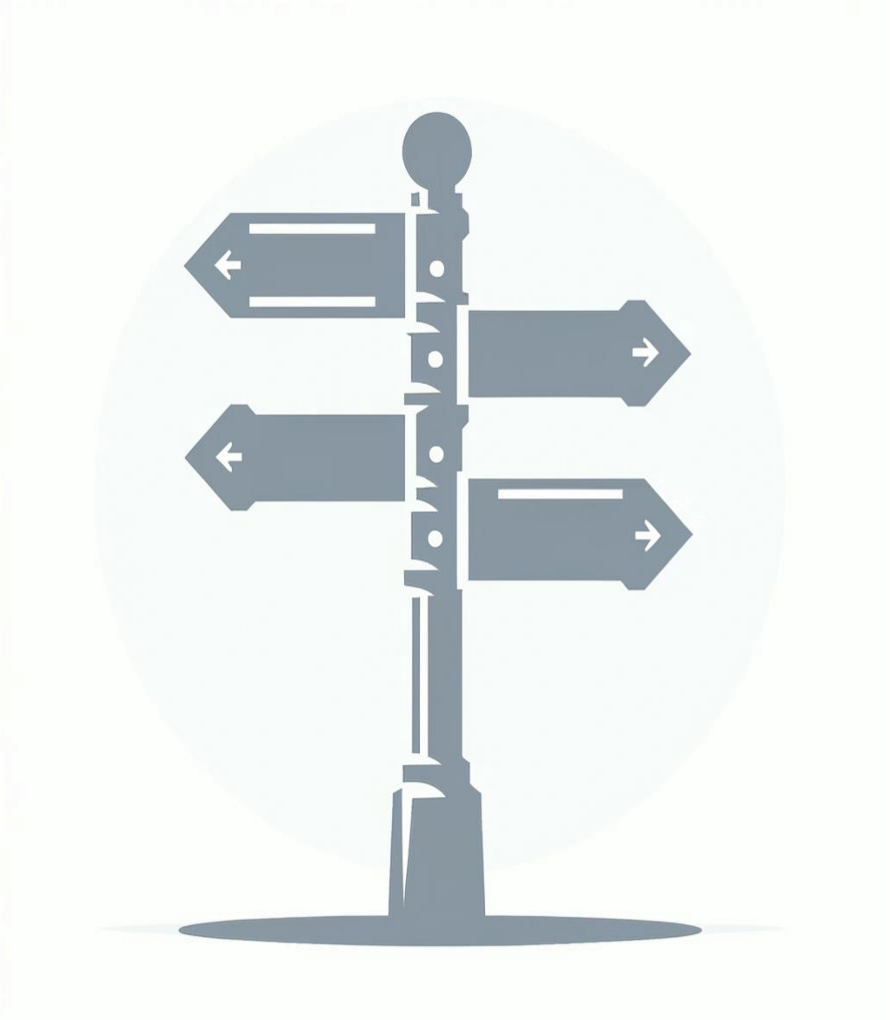


Bits $b_7 b_6 b_5$					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Row ↓ / Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	´	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

ASCII[3]

[1] and [3]: Wikipedia [2]:listverse.com

Project Scope



Global font- and typeface market 2023: ca. USD 1000 million[1]

- Font sets are created manually with takes time and efforts
- Font designers take shortcuts, i.e. not design full set

Our Project: The AI Font Generator

Prototype: AI Font Generator completes commonly missing German special characters (ä, ö, ü, ß) in style and design of original set

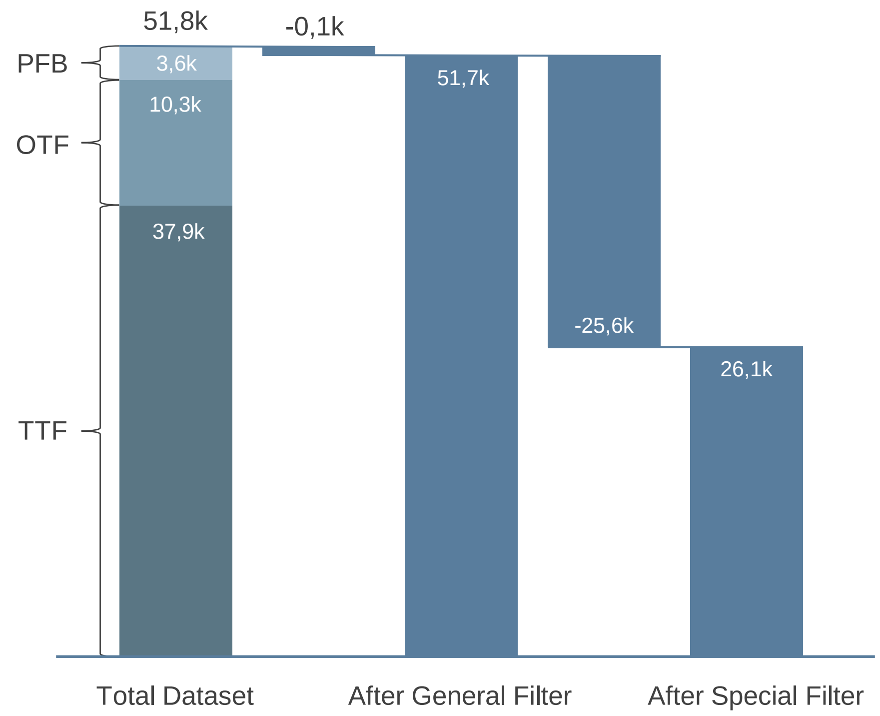
Built to be expanded. Options:

- More glyphs
- Prompt input
- A generator to create novel and complete font sets from scratch

[1] Proficient Market Insights, QYResearch Group

The Data

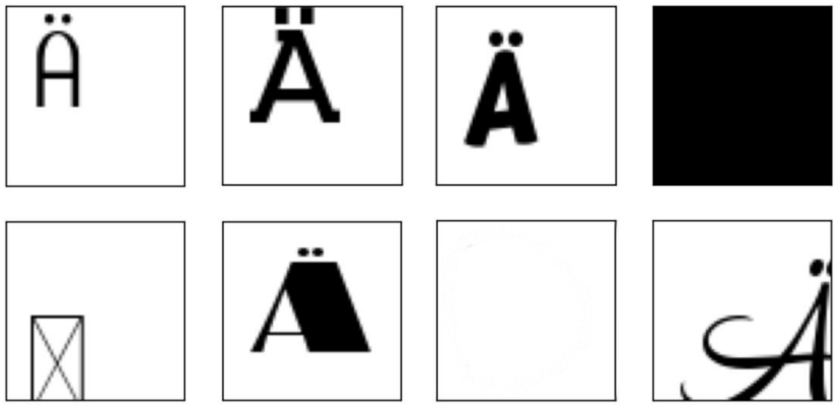
The Dataset



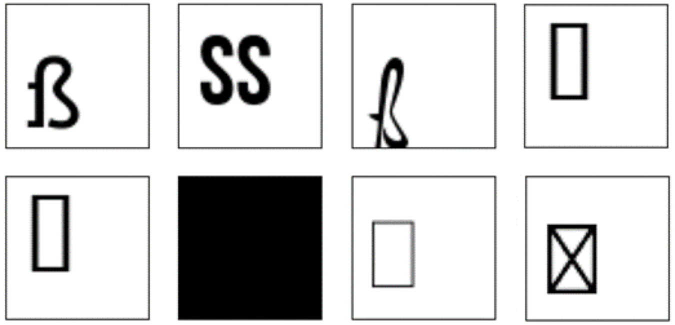
Sources: Freely available fonts from online sources

1. Conversion of PFB to TTF
2. Creation and application of several layers of filters:
   - General filter to identify and separate corrupted files
   - Specialized filters to identify font sets with glyphs that are out of bounds, empty or missing

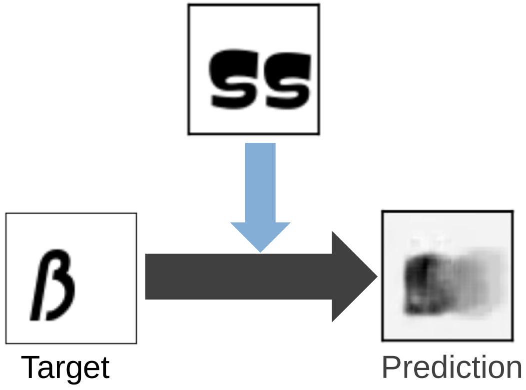
→ Nearly 50% of font sets were filtered, i.e. those not usable to German designers that need complete and correct sets



Filtering the undetectable



- Our filters detect out of bound glyphs, missing variables in cmap table and tables with cmap == None
- Our filters cannot detect placeholders/fillers in font set
- Too many font sets for manual labeling
- Those “special” special characters negatively impact training and prediction of model



Solution: Classification models to cluster symbols.

- One cluster for glyph
- One or more clusters for “Other”

Implementation: Zero-shot classification model CLIP by OpenAI

- No training and no labeling needed
- Text embedding via prompt in addition to image embedding

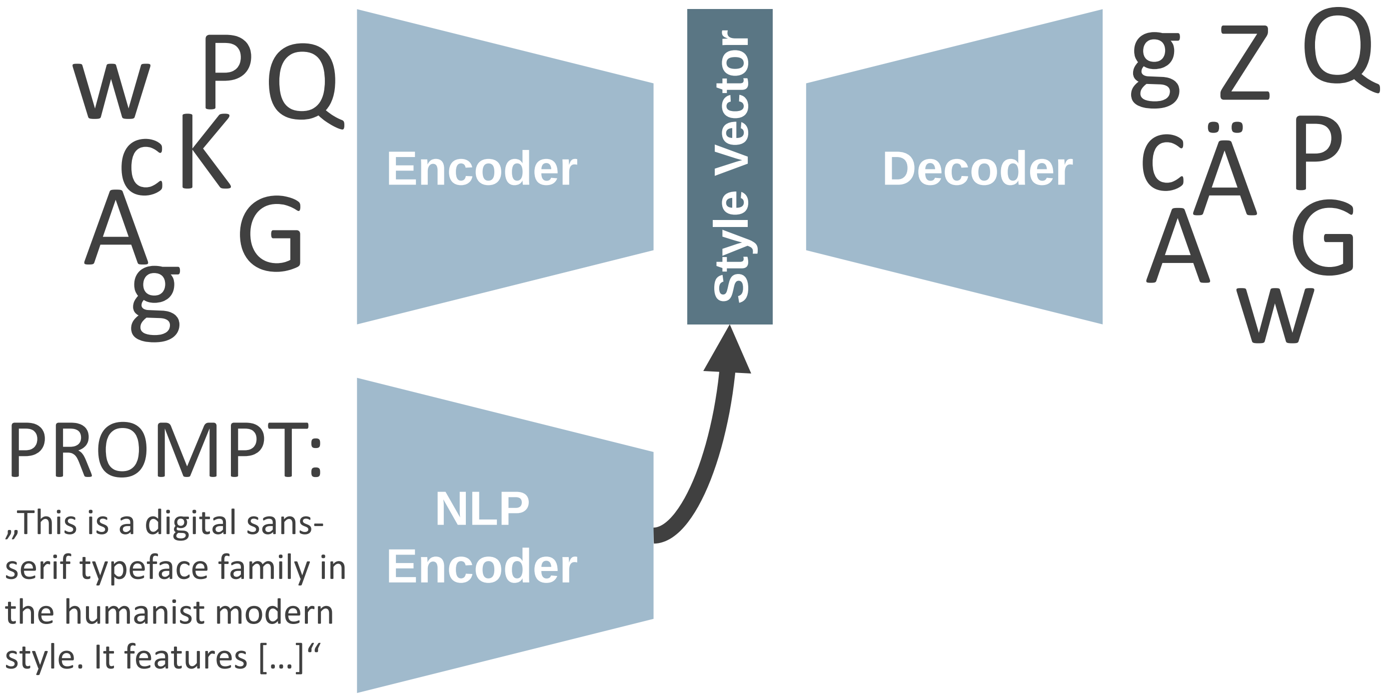
```
text_query = ['letter ß', 'letters ss', 'an X', 'one-colored box', 'rectangle', 'not letter ß']
```

- For ‘ß’: Another 5k font sets tagged

Information about fonts, file paths and filter results is stored in central Json file

The Models

Model Concepts



Objectives:

1. Generate missing characters (ä, ö, ü, ß) in same font style
2. Extendable: create novel font sets using prompt input

Encoder-Decoder structure

- Encoder: condense style
- Embedding Layer: Style representation
- Decoder: generate *all* characters
- NLP Encoder: translate human description into model language

Following two Approaches

Many Chars in, Many Chars out (MCh-MCh)

Idea: Put an incomplete array with all available characters in, get the completed array with all characters out.

Pros

- Lighter on training
  (samples = fonts, masking for Input)

Cons

- Heavier on inference

Some Chars in, One Char at a time out (SCh-1Ch)

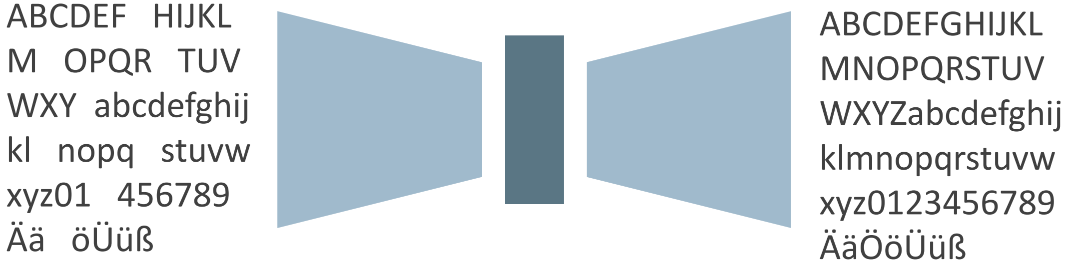
Idea: Show some characters to pretrained encoder, and tell the decoder the character to generate in the same style (One-Hot).

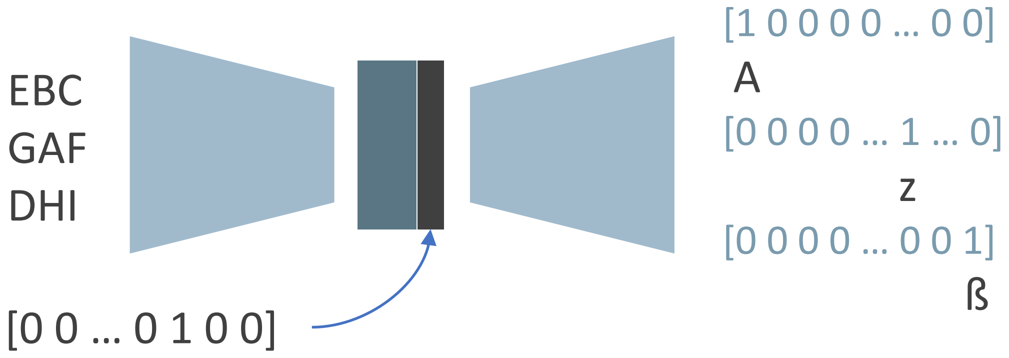
Pros

- Cheaper on inference
- Easy to teach new letters

Cons

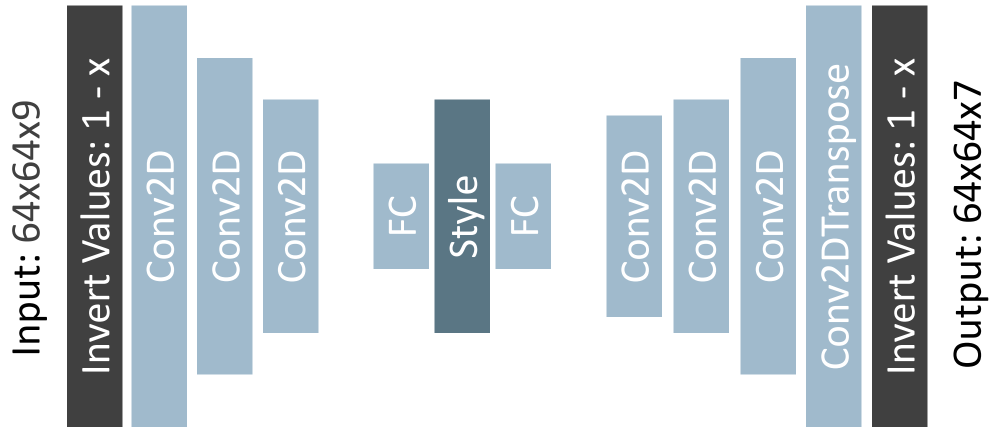
- Heavy on training (samples = fonts * characters)

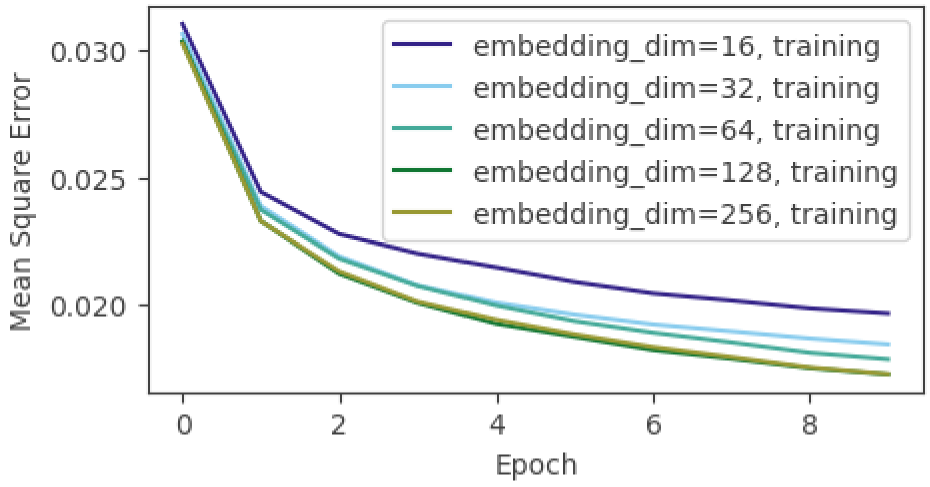




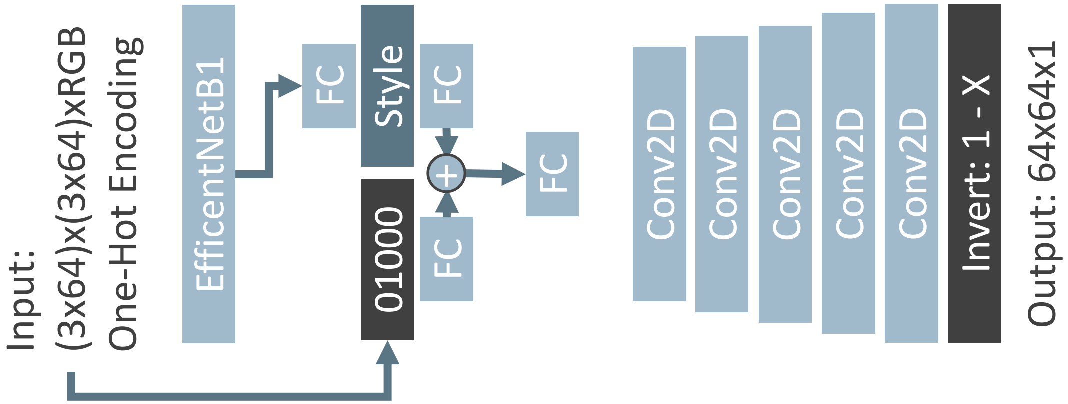
Model Architecture and Hyperparameters

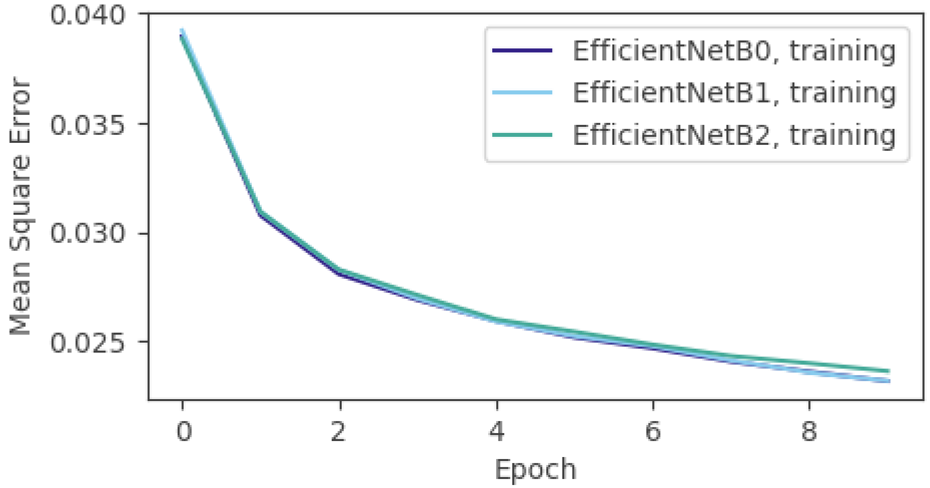
Many Chars in,
Many Chars out (MCh-MCh)



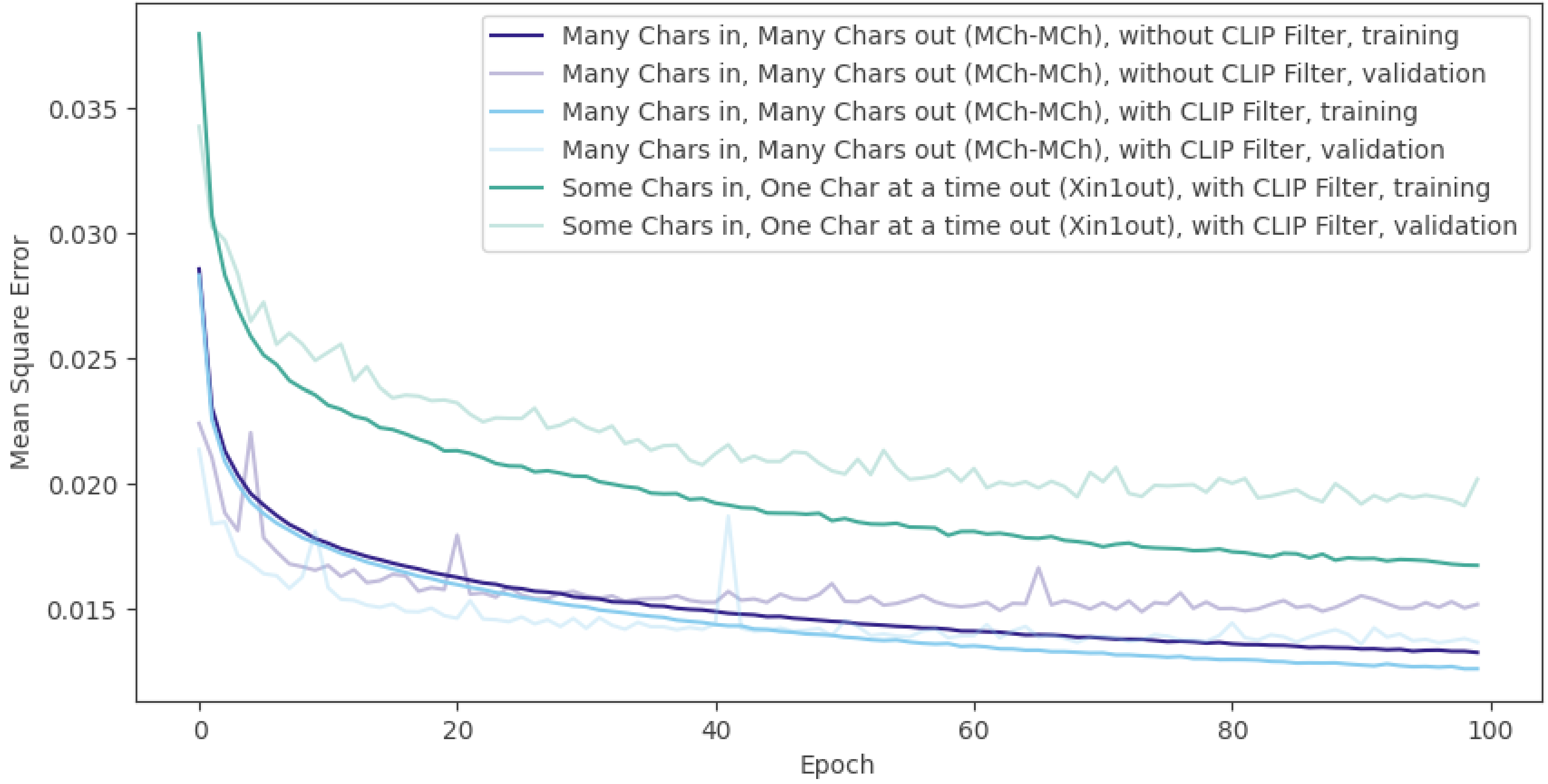


Some Chars in,
One Char at a time out (SCh-1Ch)





Model Performance



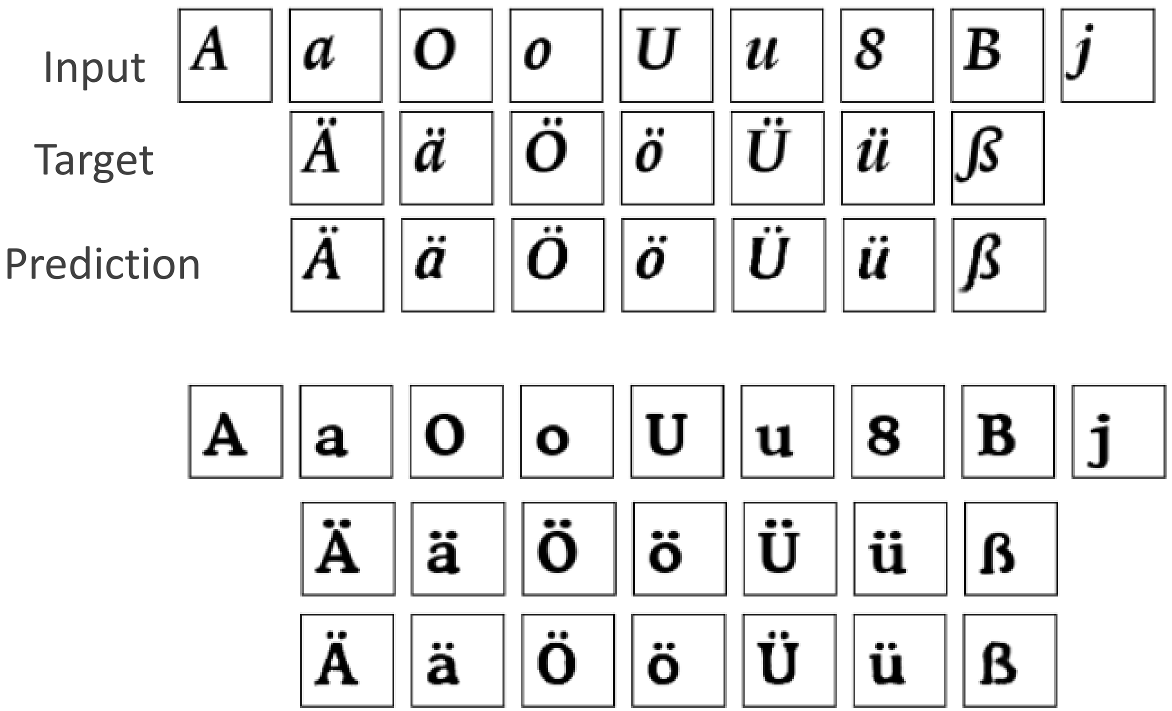
- Training for 100 Epochs
- EfficientNet with trainable weights
- Both models trained with CLIP-cleaned Data
  - Winner additionally with-out CLIP filter

- Winner: Many Char in, Many Chars out
- CLIP filter clearly improves validation loss

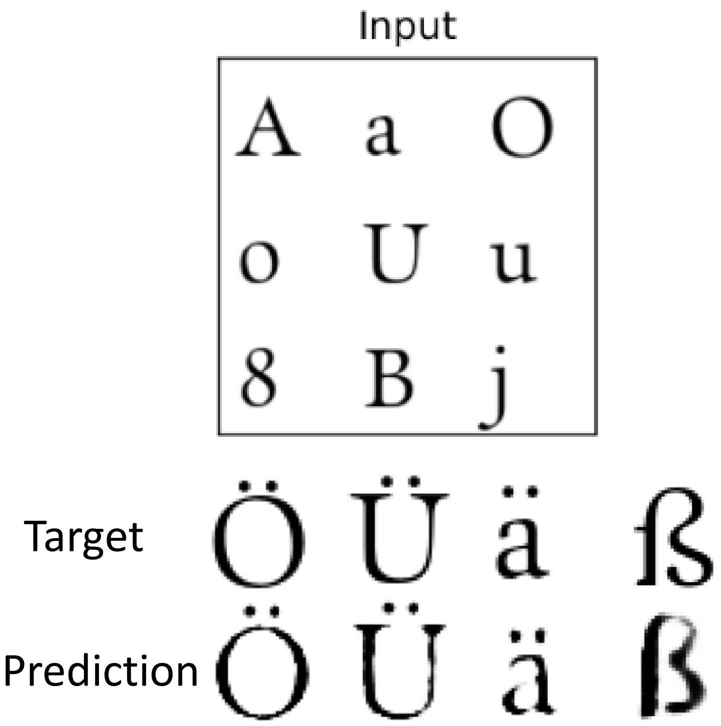
Many Chars in, Many Chars out (MCh-MCh) performs better. Best performance with applied CLIP filter to data.

Model Performance

Many Chars in,
Many Chars out (MCh-MCh)



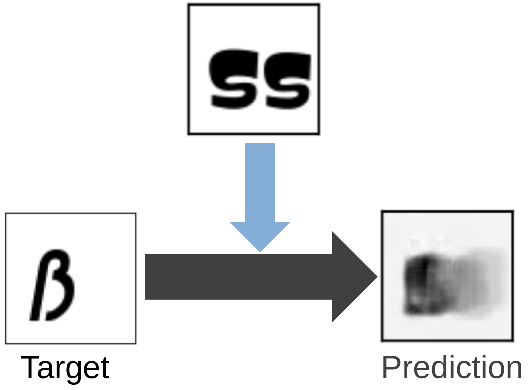
Some Chars in,
One Char at a time out (SCh-1Ch)



Decoder might learn better, if producing multiple characters at the same time.

Conclusion and Outlook

- Collected 50k Font Files
- Filtered and sorted fonts using advanced technics like CLIP
- Trained two Models capable of generating missing Characters



- Next steps
  - Convert to vector graphics and insert new characters in font files.
  - Do complete training to generate full dataset.
  - Train NLP-Encoder: Generate new fonts with prompt.