COMPRISE

Cost effective, Multilingual, Privacy-driven voice-enabled Services

www.compriseh2020.eu

Call: H2020-ICT-2018-2020
Topic: ICT-29-2018
Type of action: RIA
Grant agreement N°: 825081

WP N°4: Cost-effective multi-lingual voice interaction

Deliverable N°4.4: Final weakly supervised learning library

Lead partner: USAAR

Version No: 1.0

Date: 28/02/2021

Document information	
Deliverable N°and title:	D4.4 – Final weakly supervised learning library
Version N°:	1.0
Lead beneficiary:	USAAR
Author(s):	Thomas Kleinbauer (USAAR), David Adelani (USAAR), Ali Davody (USAAR), Imran Sheikh (INRIA)
Reviewers:	Raivis Skadiņš (TILDE) and Marc Tommasi (INRIA)
Submission date:	28/02/2021
Due date:	28/02/2021
Type[1]:	OTHER
Dissemination level[2]:	PU

Document history			
Date	Version	Author(s)	Comments
05/02/2021	0.1	Thomas Kleinbauer et al.	Initial version
25/02/2021	0.2	Thomas Kleinbauer et al.	Revised version integrating feedback and comment from internal reviewers
28/02/2021	1.0	Akira Campbell and Emmanuel Vincent	Final version revised by the Project Manager and the Coordinator

[1] **R**: Report, **DEC**: Websites, patent filling, videos; **DEM**: Demonstrator, pilot, prototype; **OTHER**: Software Tools
[2] **PU**: Public; **CO**: Confidential, only for members of the consortium (including the Commission Services)

Document Summary

This deliverable documents the advances on the COMPRISE weakly supervised learning library. Weakly supervised learning addresses the problem that state-of-the-art supervised learning methods, for many relevant tasks, require enormous amounts of manually labelled data. In this document, we explore alternatives to this costly endeavor for two specific areas of the project: Speech-to-Text and text processing.

Like its predecessor Deliverable D4.2, Deliverable D4.4 consists of two parts: a software library and this document. An overview of the different system architectures is given in Section 2. The scientific approaches underlying the software tools are detailed in Section 3. For practical applications, Section 4 gives in-depth instructions on how to install and use the library. Finally, we discuss our achievements and the potential for future extensions in Section 5.

For Speech-to-Text, we focus on training better language models while continuing with our previous idea of learning with the help of Speech-to-Text confusion networks obtained from unlabelled speech data. We present components to train n-gram as well as neural network language models from confusion networks and evaluate the underlying approaches along with the components released in the initial Deliverable D4.2 library.

For weakly supervised text processing, we present two alternatives to the approaches outlined in the initial version of this library (see Deliverable D4.2). The first approach uses Meta-Learning, a technique popular in the computer vision community and increasingly so in the natural language processing community. We applied it here to Named Entity Recognition (NER). A second approach leverages the knowledge inherent in pre-trained language models through cloze-style queries, which is applicable in few-shot and even zero-shot settings.

Furthermore, this deliverable introduces all necessary software tools for users and developers to apply our techniques. The two parts of the library can be accessed here:

- Speech-to-text:
  https://gitlab.inria.fr/comprise/speech-to-text-weakly-supervised-learning
- Text processing:
  https://gitlab.inria.fr/comprise/spoken-language-understanding-weakly-supervised-learning

Table of contents

1. Introduction

The COMPRISE project has five main objectives: privacy-by-design, inclusiveness, cost-effectiveness, sustainability, and real-world applicability. Work Package 4 (WP4) is concerned with the cost-effectiveness aspects of the project, and approaches them from two sides:

- The COMPRISE SDK (see Deliverable D4.1 "Initial COMPRISE SDK prototype", submitted to the European Commission on August 31, 2020 – Public) aims at providing an easily accessible technical infrastructure that expedites the development process of voice-enabled apps, especially for developers that are not experts in voice technologies.
- Weakly supervised learning techniques are developed to significantly reduce the necessary data annotation efforts when compared to classical supervised learning.

In other words, WP4 addresses two of the major cost factors in the development process of modern voice applications: the time and cost to bootstrap a basic app, and the time and cost to employ state-of-the-art machine learning technologies.

This Deliverable D4.4 concerns itself with the weakly supervised parts of WP4, and is an extension to Deliverable D4.2 "Initial weakly supervised learning library" (Submitted to the European Commission on April 29, 2020 – Public). Like its predecessor, this deliverable consists of two parts, this document and a software library. Both parts address Speech-to-Text (STT), Spoken Language Understanding (SLU), and Dialogue Management (DM).

Since the submission of Deliverable D4.2, the approaches described therein have been extended or even superseded where alternative developments in the research community towards the same goal have gained traction. In particular, we have reported in Deliverable D4.2 about alternatives to weakly supervised approaches for SLU and DM. Despite showing considerable improvements over the supervised baseline, the classification rates achieved by our weak supervision approach still left to be desired for practical application. We therefore extended the work begun at the end of the first reporting period and focused on domain adaptation approaches which seemed to be more promising. In fact, this turned out to be a good decision, as we are now able to reach much better classification results for few-shot or even zero-shot settings, translating to a greatly reduced need for costly annotation.

Deliverable D4.2 provided two STT components, representing two distinct approaches, for training STT models: (1) semi-supervised training driven by error detection (Err2Unk), and (2) dialogue state-based weakly supervised training. The first method trained STT models by guiding a state-of-the-art Acoustic Model (AM) training method with error predictions inferred from STT confusion networks. The second method exploited weak supervision from utterance-level dialogue state labels. These components focused on obtaining reliable transcriptions from unlabelled speech data which could be used for training both STT AMs and Language Models (LMs). The two components can be used independently or combined together. Moreover, our evaluation on different limited data setups showed that the Err2Unk training component resulted

in significant improvements over the state-of-the-art methods and gave additional improvements when used along with weak supervision from dialogue states.

This deliverable builds on top of the Err2Unk approach and provides a dedicated component for training STT LMs, including statistical n-gram LMs and Recurrent Neural Network (RNN) based LMs. As compared to existing STT LM training tools, the COMPRISE STT LM component is aimed at learning LMs from alternate and uncertain STT hypotheses, obtained from unlabelled speech data using initial or domain-mismatched STT models.

This document gives an overview over the scientific approaches for both tasks and documents the open-source releases of the accompanying software tools. Section 2 gives a quick overview of the design of the STT training components after including the new COMPRISE STT LM component, and of the training component for text processing. The underlying scientific approaches and the corresponding software tools are presented in Sections 3 and 4, respectively. We conclude in Section 5.

2. Design and implementation of the learning components

2.1. Learning components for Speech-to-Text

The final weakly supervised learning library of COMPRISE retains the two main learning components for STT that were provided as part of the initial library. It adds a new STT LM training component which enables the learning of LMs from STT confusion networks. Hence, it is named as Confusion Network based LM (CN2LM). While the initial library also relied on confusion networks for Err2Unk training, the CN2LM training component can be used independently or along with the Err2Unk component.[3]

Figure 1 presents an overview of the typical usage of the Err2Unk and CN2LM training components for learning STT models. The implementation relies on the Kaldi STT toolkit,[4] as highlighted in blue. The main contributions through the COMPRISE libraries are highlighted in red and green colours, with red representing the Err2Unk component from the initial library and green representing the CN2LM training component. Additionally, Table 1 presents a quick reference on the implementation of these two components.

2.2. Learning components for text processing

Concerning text processing, a graphical depiction of the full method is shown in Figure 2. We focus on Named Entity Recognition using a pre-trained LM (BERT), a template, and lists of words representative of each Named Entity. The approach is exemplified here with a concrete example. Details of the underlying approach are given in the following chapter.

[3] Similarly, the CN2LM training component can also be used along with dialogue state driven weakly supervised training of STT models, that was provided in the initial library.

[4] `http://kaldi-asr.org/`

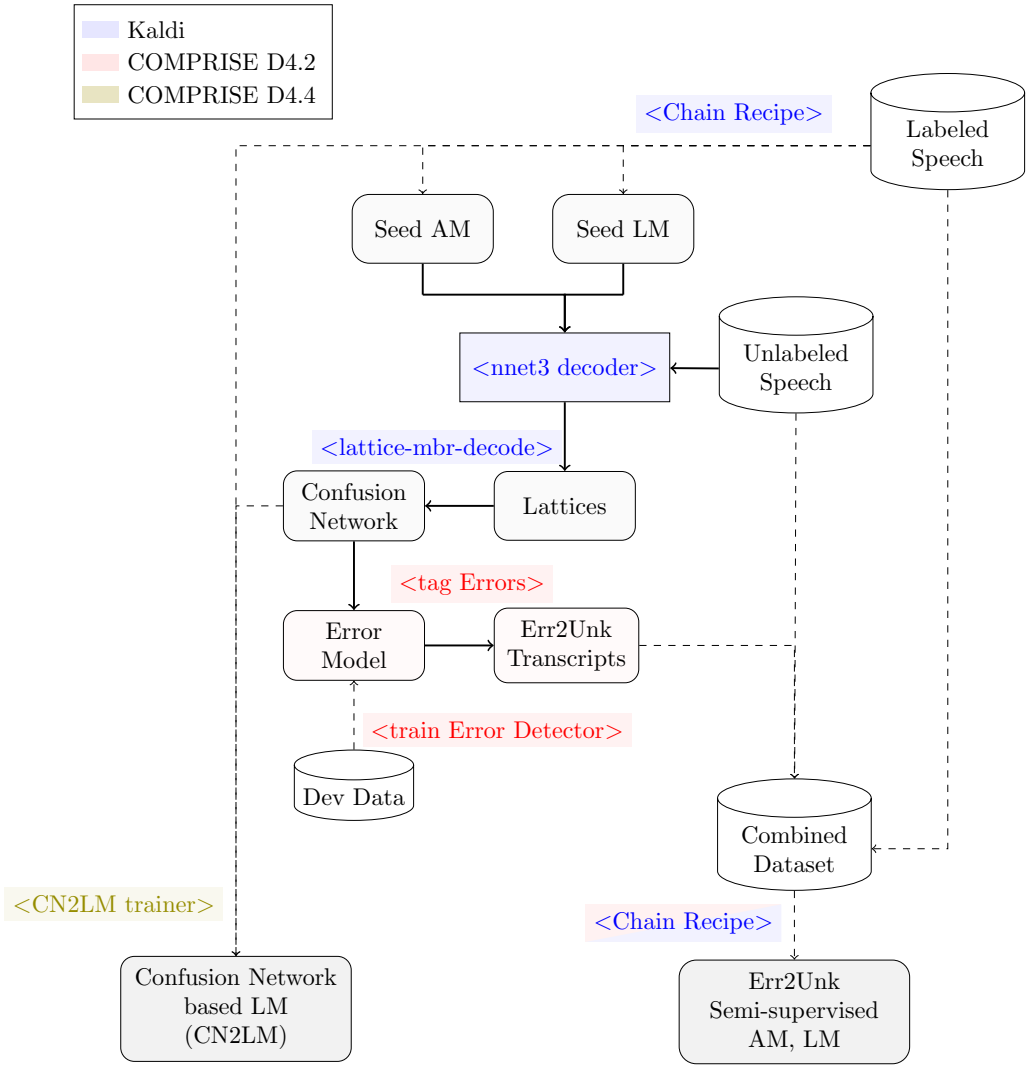

Figure 1: Overview of error detection (Err2Unk) and confusion network (CN2LM) driven training of STT models (AM, LM). (Dashed arrows indicate 'use for training'.)

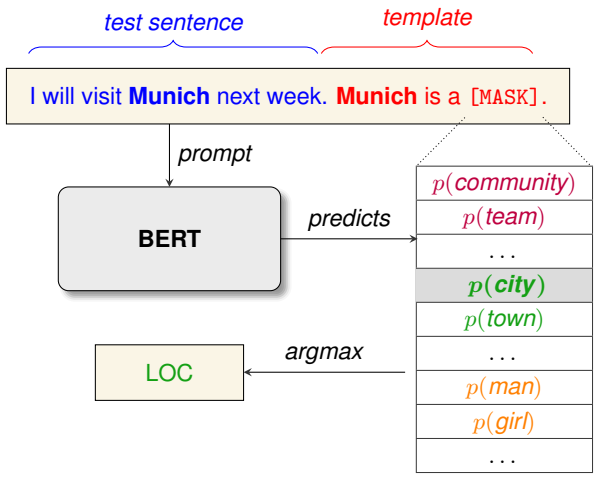

Figure 2: System setup at test time. The input prompt to the BERT model is the concatenation of the test sentence and the template, with the token to classify (here: *Munich*) pre-inserted into the template. The output (*LOC*) is the predicted entity label for that token.

Table 1: Overview of error detection (Err2Unk) and confusion network (CN2LM) driven training of STT models. (Note: Steps 3–5 are optional and are only for training a new LM, Step 6 is optional and is only for training a new AM.)

Step	Description	Inputs	Outputs	Implementation notes
1	Train seed STT models	Labelled speech	Seed AM, LM	Kaldi Chain recipe: Bash scripts with Kaldi binaries for processing speech, Perl scripts to manage inputs and train LM, Python scripts to train AM.
2	Prepare Confusion Networks	Unlabelled speech, dev data	Confusion networks	Kaldi binaries to decode speech to lattices, convert lattice to confusion network
3	Train Error Detector	Dev confusion networks	Error detection model	COMPRISE libraries: Python scripts to extract features, train error model
4	Obtain unlabelled speech transcripts	Unlabelled speech confusion networks	Speech transcripts	COMPRISE libraries: Python scripts to tag errors, format transcripts
5	Retrain AM	Combined data	New AM	Kaldi Chain recipe (as Step 1)
6	Retrain LM	Labelled speech transcripts, unlabelled speech confusion networks	New LM	COMPRISE libraries: Python scripts to train n-gram and RNN LM

3. Scientific approach

3.1. Cost-effective learning for Speech-to-Text

3.1.1. Learning with Speech-to-Text errors and confusion

The initial COMPRISE weakly supervised learning library presented the STT confusion network and error detection driven approach for semi-supervised training of STT models. The focus of the underlying approach was to obtain reliable STT transcriptions on the unlabelled speech for training STT AMs and LMs. The final weakly supervised learning library presents CN2LM: a confusion network based LM learning component.

As shown in Figure 3, STT confusion networks contain a sequence of confusion bins with each bin containing one or more arcs representing alternative word hypotheses. Each arc in a bin has an associated posterior probability or score, denoting that one word hypothesis is more likely than others. The main motivation of CN2LM is to exploit the alternative hypotheses and uncertainty embedded in the confusion network to learn LMs from unlabelled speech data. The CN2LM component features training of 3-gram LM as well as RNN LM. We briefly discuss our approach to train these two types of LMs from STT confusion networks.

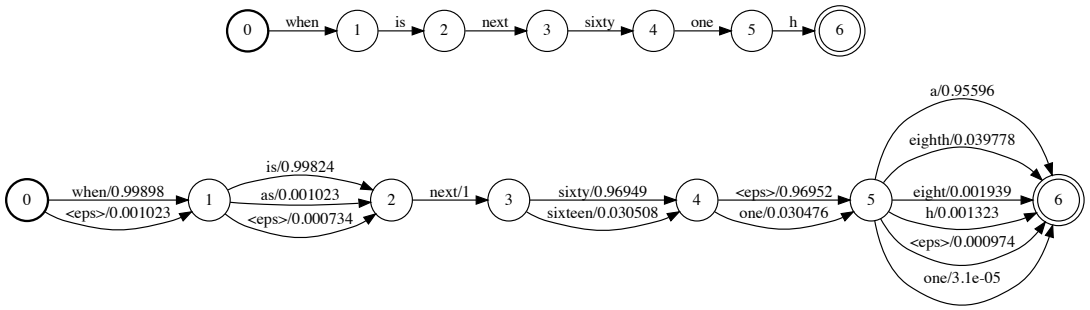

Figure 3: Graphical representation of the reference transcription (top) and the STT decoded confusion network (bottom).

3.1.1.1. Learning n-gram language models If we denote n-gram word sequences occurring in a text corpus by $\mathbf{u}w$, where $\mathbf{u}$ represents the context of previous $n-1$ words for word w, then an estimate of n-gram LM probabilities can be obtained as

$$p(w|\mathbf{u}) = \frac{c(\mathbf{u}w)}{\sum_w c(\mathbf{u}w)} \tag{1}$$

with $c(.)$ denoting counts. Smoothing methods need to be applied to these simple count based estimates in order to obtain practically usable LMs. Modified interpolated Kneser-Ney (KN) smoothing is known to provide the best performing n-gram LMs.[5]

However, arcs or words appearing in STT confusion networks carry fractional weights or scores and KN smoothing cannot be applied directly. A modified interpolated *expected* KN smoothing (ieKN) approach was proposed to handle fractional counts[6] and recently applied to learn n-gram LMs from crowdsourced and STT transcriptions.[7] We extend the ieKN approach to STT confusion networks to learn n-gram LMs. Our approach first extracts n-gram bin sequences from the confusion network and then populates different possible word sequences of length n. Each word sequence is assigned a score by multiplying the associated arc posteriors. The ieKN smoothing approach is applied to obtain the n-th order probability estimates and a recursive smoothing technique, similar to KN smoothing, is applied to obtain the lower order probability estimates.

In terms of computational complexity, ieKN smoothing is identical to the traditional KN smoothing. Additional computation is required by ieKN smoothing for estimating expected counts from the fractional weights/scores of the word sequences of length n. This computation is linear in terms of the number of distinct word sequences of length n. Moreover, assigning scores to the word sequences observed in the STT confusion network requires $O(TA^n)$ computations, where A is the average number of arcs in a confusion bin and T is total number of confusion bins observed in the unlabelled set. In practice, $n = 3$, $A \approx 3$ and T is equivalent to the total number of words expected in the unlabelled dataset.

3.1.1.2. Learning neural network language models Given a sequence of word-level tokens $w_1, w_2, ...w_t, ..., w_N$ from a corpus, a word-based RNN language model with

[5] Stanley F. Chen and Joshua Goodman. "An Empirical Study of Smoothing Techniques for Language Modeling". In: *34th Annual Meeting of the Association for Computational Linguistics*. June 1996, pp. 310–318.

[6] Hui Zhang and David Chiang. "Kneser-Ney Smoothing on Expected Counts". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. June 2014, pp. 765–774.

[7] Michael Levit, Sarangarajan Parthasarathy, and Shuangyu Chang. "What to Expect from Expected Kneser-Ney Smoothing". In: *Proc. Interspeech 2018*. 2018, pp. 3378–3382.

parameters $\theta^w, \theta^h, \theta^y$ relies on the following formulation:

$$h_t = \sigma(\theta^h \; h_{t-1} + \theta^w \; w_t) \tag{2}$$

$$q(\hat{w}_{t+1}|h_t) = \text{softmax}(\theta^y \; h_t) \tag{3}$$

where h_t represents the RNN hidden state which tries to model the history or context observed until w_t, $\sigma(.)$ is a nonlinear activation function, and the softmax function estimates the history-dependent word level LM probabilities $q(\hat{w}_{t+1}|h_t)$. The RNN LM parameters $\Theta = \{\theta^w, \theta^h, \theta^y\}$ are trained to minimise the loss function:

$$\widehat{\Theta} \approx \arg\min_{\Theta} \sum_{h} \sum_{j} -\log \; q(\hat{w}^j_{t+1}|h_t). \tag{4}$$

State-of-the-art STT systems include RNN LMs to rescore the STT outputs decoded by the AM and an n-gram LM. However, typical RNN LMs cannot be applied readily to STT confusion networks. As depicted in Figure 3, each t in a confusion network corresponds to a confusion bin with different alternative word hypotheses. This implies that Equations (3) and (4) cannot be applied. In order to address this, a hidden state h^i_t can be first computed for each possible arc w^i_t in a bin and then a pooled state can be obtained as follows:

$$h^i_t = \sigma(\theta^h \; h_{t-1} + \theta^w \; w^i_t) \tag{5}$$

$$h_t = \text{pool}_i(h^i_t) \tag{6}$$

where a standard mean, average, max or even attention based pooling mechanism can be used. Additionally, the loss function must be updated to take care of the multiple output arcs w^j_{t+1} possible at the next step $t+1$. To address this, we compute the Kullback-Leibler (KL) divergence between the RNN predictions and the confusion bin posteriors $p(w^j_{t+1}|o_t)$. The LM loss function is then formulated as:

$$\widehat{\Theta} = \arg\min_{\Theta} \sum_{t} \underset{j \in V}{\text{KL}} \left(p(w^j_{t+1}|o_{t+1}), q(\hat{w}^j_{t+1}|h_t) \right) \tag{7}$$

where V denotes the LM vocabulary and o_{t+1} denotes the observed speech signal which leads to the confusion bin posteriors.

Applying RNN LMs on STT confusion networks may appear to increase the computational complexity of the LM. However, it can be shown that the order of computational complexity remains similar to that of an RNN LM applied to a text transcription of the same length. Consider an RNN LM with vocabulary V being trained on a collection of text transcriptions having a total length of T words. Equations (2) and (3) contribute the main computations in the forward pass of the RNN. If we denote the number of hidden layer neurons as H then the order of computations in the forward pass can be given as:

$$\begin{aligned} f(T, V, H) &\approx (HH + HV)T \\ &= O(HVT) \; : \; H \ll V \end{aligned} \tag{8}$$

The backward pass of the RNN sees a similar order of computations. In case of an RNN LM over STT confusion networks, Equation (5) brings additional computation. If we consider that each confusion bin has on average A arcs then the order of computations in the forward pass can be given as:

$$\begin{aligned} f(T, V, H, A) &\approx (HHA + HV)T \\ &= O(HVT) \; : \; H \ll V, HA \ll V \end{aligned} \tag{9}$$

3.1.2. Experiments and evaluation

3.1.2.1. Experimental setup We investigate STT improvements in two different scenarios detailed below. Table 2 briefly presents the different datasets and the splits used in these evaluation setups.

Human-human conversations: This is a matched domain limited data scenario wherein both the labelled and unlabelled data are taken from the Verbmobil corpus.[8] We have divided the entire Verbmobil English speech corpus into four splits and ensured that there are no overlapping speakers or conversations across the four splits. Based on the labelled and unlabelled training splits, we denote this setup as VM5-VM20.

Human-machine dialogue: We also analyse the matched domain scenario wherein both the labelled and unlabelled data are human utterances extracted from a human-machine dialogue system. We use subsets of the annotated Let's Go dataset[9] to emulate this setup. We use data corresponding to the first 15 days of October 2008 as the labelled dataset and speech corresponding to the next two and half months as unlabelled data. Data corresponding to the first 15 days and the last 15 days of September 2009 are treated as development and test sets, respectively. We denote this setup as LG4-LG19.

Table 2: Datasets and splits used in the evaluation setup. VM = Verbmobil, LG = Let's Go, h = hours.

	VM5-VM20	LG4-LG19
Train labelled	VM 5 h	LG 4 h
Train unlabelled	VM 20 h	LG 19 h
Development	VM 2 h	LG 6 h
Test	VM 3 h	LG 6 h

3.1.2.2. Evaluation In order to evaluate the CN2LM approaches for learning LMs from confusion networks, we report the standard Perplexity and Word Error Rate (WER) measures on the development (DEV) and test (TEST) sets of the VM5-VM20 and LG4-LG19 setups. Our evaluation compares the following systems:

- Seed: the seed models trained only on the labelled sets (VM5 or LG4).
- Bestpath: semi-supervised models obtained in combination with the labelled set and best-path STT transcriptions obtained on the unlabelled speech using the seed models.
- CN2LM: semi-supervised n-gram or RNN LM trained on reference transcriptions of the labelled set and confusion networks obtained by seed models on the unlabelled speech.
- Err2Unk: semi-supervised models obtained in combination with the labelled set and Err2Unk transcriptions obtained on the unlabelled speech using the seed models and error prediction.

[8] Susanne Burger et al. "Verbmobil Data Collection and Annotation". In: *Verbmobil: Foundations of Speech-to-Speech Translation*. Ed. by Wolfgang Wahlster. 2000, pp. 537–549.

[9] DailRC. *The Integral LET'S GO! Dataset*. Last accessed April 1, 2020. URL: https://dialrc.github.io/LetsGoDataset/.

- Oracle: models trained on labelled and unlabelled data along with their reference transcriptions.

The systems rely on the Kaldi Chain AM with Time Delay Neural Network (TDNN) layers and i-vectors for speaker adaptation. Due to the limited data setups, our evaluation uses 3-gram LM and Gated Recurrent Unit (GRU) based single layer RNN LM. Following the standard approach, the 3-gram LM is used for first pass STT decoding and RNN LM for pruned lattice rescoring.[10]

Table 3 presents a perplexity evaluation of the seed and semi-supervised 3-gram LMs on the development and test sets of the two setups. It shows that there are no significant differences in the LG4-LG19 setup which deals with short queries related to bus schedule information. However, for the VM5-VM20 conversational setup, CN2LM 3-gram LM achieves significant perplexity improvements over the Bestpath semi-supervised LM. Err2Unk semi-supervised LM shows higher perplexity due to high <unk> LM probabilities, after retaining all error regions as <unk> in the unlabelled speech transcripts.

Table 3: Perplexity of semi-supervised 3-gram LM. (Bold font denotes lowest perplexity.)

	Seed LM		Bestpath Semi-sup LM		Err2Unk Semi-sup LM		CN2LM Semi-sup LM	
	DEV	TEST	DEV	TEST	DEV	TEST	DEV	TEST
VM5-VM20	77.32	78.04	62.97	66.61	67.58	70.85	**61.61**	**64.82**
LG4-LG19	14.53	13.70	**14.34**	**13.40**	15.39	14.43	14.49	13.42

Table 4 presents a perplexity evaluation of the Bestpath and CN2LM based RNN LMs. The improvements in the LG4-LG19 setup are not significant. However, the VM5-VM20 conversation setup sees a big reduction in perplexities.

Table 4: Perplexity of semi-supervised RNN LM. (Bold font denotes lowest perplexity.)

	Bestpath Semi-sup RNN		CN2LM Semi-sup RNN	
	DEV	TEST	DEV	TEST
VM5-VM20	62.67	67.29	**56.14**	**59.84**
LG4-LG19	14.64	13.49	**14.35**	**13.24**

WER results on the development and test sets of the VM5-VM20 setup are presented in Tables 5 and 6. Table 5 shows the effect of different AM and LM combinations during the first pass decoding with 3-gram LMs. Seed and Oracle model performances highlight the upper and lower bounds of the WER, respectively. Table 5 shows that Err2Unk AM+LM gives the lowest WER and outperforms the traditional Bestpath semi-supervised AM+LM system. However, we would like to highlight that the CN2LM semi-supervised 3-gram LM achieves lower WER compared to the Bestpath semi-supervised LM. Table 6 shows that minor WER reduction can also obtained with CN2LM semi-supervised RNN LM.

WER results on the development and test sets of the LG4-LG19 setup are presented in Tables 7 and 8. Similar to the VM5-VM20 setup, WER evaluation on the LG4-LG19 setup shows that the Err2Unk semi-supervised AM+LM achieves the lowest WER.

[10] H. Xu et al. "A Pruned Rnnlm Lattice-Rescoring Algorithm for Automatic Speech Recognition". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5929–5933.

Table 5: WER of semi-supervised AMs and 3-gram LMs in the VM5-VM20 setup. (Bestpath LM column represents Oracle LM for Oracle AM row. Bold font denotes lowest WER and underline denotes the difference is not statistically significant compared to the lowest WER. * denotes CN2LM WER differences are statistically significant compared to respective Bestpath LM.)

	Seed LM		Bestpath Semi-sup LM		Err2Unk Semi-sup LM		CN2LM Semi-sup LM	
	DEV	TEST	DEV	TEST	DEV	TEST	DEV	TEST
Seed AM	39.52	39.77	-	-	-	-	-	-
Bestpath Semi-sup AM	33.91	34.35	31.88	32.90	30.08	30.86	32.08	32.56*
Err2Unk Semi-sup AM	30.74	32.12	29.15	30.70	**28.66**	**29.74**	29.60	30.38*
Oracle AM	26.12	26.42	21.75	22.20	-	-	-	-

Table 6: WER after RNN LM re-scoring in the VM5-VM20 setup. (Bold font denotes lowest WER and underline denotes the difference is not statistically significant compared to the lowest WER. * denotes CN2LM RNN WER differences are statistically significant compared to Bestpath RNN.)

	Bestpath Semi-sup RNN		CN2LM Semi-sup RNN	
	DEV	TEST	DEV	TEST
Bestpath Semi-sup AM+LM	30.76	32.00	30.76	31.59*
Err2Unk Semi-sup AM+LM	27.64	28.95	**27.82**	**28.82**

Table 7: WER of semi-supervised AMs and 3-gram LMs in the LG4-LG19 setup. (Bestpath LM column represents Oracle LM for Oracle AM row. Bold font denotes lowest WER and underline denotes the difference is not statistically significant compared to the lowest WER.)

	Seed LM		Bestpath Semi-sup LM		Err2Unk Semi-sup LM		CN2LM Semi-sup LM	
	DEV	TEST	DEV	TEST	DEV	TEST	DEV	TEST
Seed AM	38.98	38.46	-	-	-	-	-	-
Bestpath Semi-sup AM	34.82	33.72	34.53	33.13	33.96	32.63	34.29	33.14
Err2Unk Semi-sup AM	32.39	31.32	31.99	30.66	**31.90**	**30.44**	31.97	30.52
Oracle AM	32.56	30.87	31.76	29.57	-	-	-	-

Table 8: WER after RNN LM re-scoring in the LG4-LG19 setup. (Bold font denotes lowest WER and underline denotes the difference is not statistically significant compared to the lowest WER.)

	Bestpath Semi-sup RNN		CN2LM Semi-sup RNN	
	DEV	TEST	DEV	TEST
Bestpath Semi-sup AM+LM	33.95	32.64	34.30	32.96
Err2Unk Semi-sup AM+LM	31.64	30.11	**31.63**	**29.91**

However, Err2Unk AM with Bestpath and CN2LM based semi-supervised 3-gram LM also achieve a similar level of performance. Similarly, rescoring with Bestpath or CN2LM based semi-supervised RNN LMs reduces overall WER but does not show a significant difference among the two approaches.

Overall it can be concluded that, for the Verbmobil dataset, that contains more spontaneous conversations, CN2LM based 3-gram LM gives consistent perplexity and WER

reductions over Bestpath semi-supervision. CN2LM based RNN LM gives large perplexity reductions but this did not translate into a WER reduction. In the case of the Let's Go dataset, that contains more restricted and shorter queries, differences in perplexity and WER performance of CN2LM based LMs and Bestpath LMs are not statistically significant. However, it must be noted that most improvements in the Let's Go setup come from the Err2Unk AM, which results in a performance level close to that of the oracle system. Interestingly, Err2Unk AM+LM gives the lowest WER performance in both the Verbmobil and Let's Go setups, with 3-gram LM decoding as well as RNN LM rescoring.

3.2. Cost-effective learning for text processing

The general situation that users of SLU and DM components find themselves in has been described in Section 3.2 of Deliverable D4.2, and shall be summarised again here to remind the reader of the context of our work.

The state of the art in many tasks in Natural Language Processing (NLP) relies on Machine Learning, oftentimes in the form of supervised learning. Applying such approaches to new domains requires developers to perform two main preparatory steps:

1. Create a sufficiently large collection of in-domain data.
2. Annotate the collected data with the desired labels.

Deep learning approaches in particular give rise to a dilemma: on the one hand, they typically outperform competing approaches in terms of prediction accuracy; on the other hand, they work best with particularly large amounts of training data. This is exactly where the cost aspect becomes relevant: both of the above steps potentially take a long time, require a substantial effort, and draw on costly resources. Especially small and medium size enterprises might thus shy away from using supervised learning because they cannot justify this significant investment.

As discussed in Deliverable D4.2, while collecting enough data to successfully train a Deep Learning algorithm might be prohibitive, it may be feasible, even for smaller companies, to collect and annotate a smaller amount of data. As we've demonstrated before, this can then be used to apply techniques such as weakly supervised learning to build models that supersede classic supervised algorithms trained on the same small data. However, while significantly outperforming the supervised baselines in our earlier work, the absolute performance of our initial weakly supervised learning setup still left to be desired. This motivated the consideration of alternative approaches towards the same cost-effectiveness goal, which we will discuss in the following.

3.2.1. Weak supervision and alternatives

In COMPRISE, we have so far considered three alternative approaches to cost-effective NLP. Weak supervision in its core form has already been presented in Deliverable D4.2 but we start with a summary of the approach previously taken for comparison and context. After that, we describe a number of experiments we have run using Meta-Learning, an approach popular in Computer Vision and also increasingly so in NLP. Unfortunately, this line of research did not achieve the desired outcomes, either. Lastly, we describe a third approach that applies pre-trained LMs to allow for

few-shot or even zero-shot classification and yields very competitive results, leading to both good performance and reduced annotation costs. For that reason, it is the approach we ultimately settled on for the software portion of this deliverable.

3.2.1.1. Weak supervision Our weak supervision approach as described in Deliverable D4.2 started out with the premise that besides high-quality (but costly) manual annotations, it is often feasible for different tasks to create annotations more affordably with the help of automatic labellers, with the downside that the resulting annotations are *noisy*.

This fact was appreciated through using a neural network architecture that extended a base network by an additional noise layer (see Figure 4). The rationale was that during training, a small portion of manually labelled high-quality data was interspersed with a large amount of automatically labelled noisy data, and the additional layer learned how to de-noise the predictions of the network.

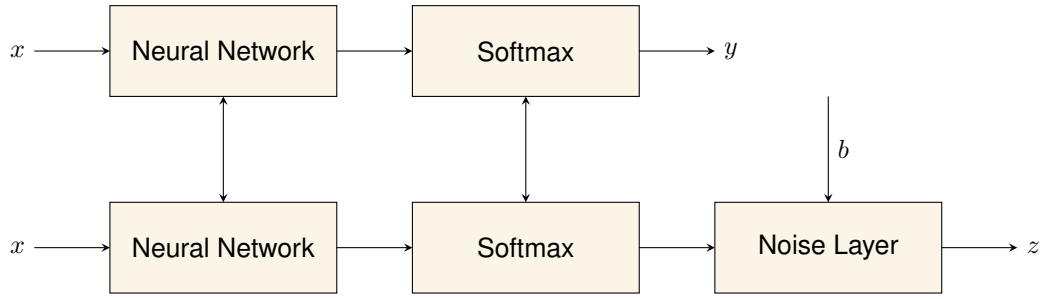

Figure 4: Illustration of weakly supervised learning.

We explored different models for generating noisy annotations, and found that the best performance was achieved with a sampling method that takes the part-of-speech of the word to be relabelled as well as the the label of the preceding word into account. However, when using 1,773 ($= 1\%$) clean samples from the CoNLL dataset for Named Entity Recognition, even that sampling method only achieves an F1-score of 42.77%. This score significantly outperforms the baseline (using just the clean samples in a supervised fashion) of 33.07%. However, an F1-score of 42.77% is still too low to be of practical use, which motivated our consideration of alternative approaches.

3.2.1.2. Meta-Learning Meta-learning or *learning to learn*[11] is a popular approach to few-shot learning (i.e., when only a few labelled examples are available in the target domain) with many applications in computer vision[12] and some applications in NLP (see[13] for an overview). In the context of NER, most applications of meta-learning for

[11] S. Ravi and H. Larochelle. "Optimization as a Model for Few-Shot Learning". In: *ICLR*. 2017; Chelsea Finn, P. Abbeel, and S. Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *ICML*. 2017; Jake Snell, Kevin Swersky, and Richard Zemel. "Prototypical Networks for Few-shot Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 4077–4087. URL: https://proceedings.neurips.cc/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf.

[12] Timothy M. Hospedales et al. "Meta-Learning in Neural Networks: A Survey". In: *ArXiv* abs/2004.05439 (2020); Eleni Triantafillou et al. "Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=rkgAGAVKPr.

[13] Wenpeng Yin. "Meta-learning for Few-shot Natural Language Processing: A Survey". In: *ArXiv* abs/2007.09604 (2020).

few-shot make use of Prototypical Networks[14] or Model-Agnostic Meta-Learning.[15] All these approaches require training on diverse domains or datasets to generalise to new domains.

The goal of Model-Agnostic Meta-Learning (MAML) is to train a model on diverse learning tasks such that it can perform well on new tasks with only a small number of examples. In our case, different domains would take the role of these tasks. This approach can be used for both zero-shot and few-shot learning. In normal deep learning training, we learn a single set of network parameters θ on all available domains. However, in MAML, the key idea is to split the training data into two, and learn two sets of network parameters θ and θ'. θ' is learned by taking gradient descent over the loss of the first-half of the training samples, while θ is learned using the average loss on the second-half of the training ground-truth labels and predictions obtained by θ'. This helps to model how the final model θ will perform on new domains.

Table 9 shows the result for zero-shot domain adaptation for NER on the OntoNotes 5.0 dataset.[16] This dataset has six domains with annotation: broadcast conversation (BC), broadcast news (BN), magazines (MZ), newswire (NW), telephone conversations (TC), and web blogs (WB). For all experiments, we use trained a BiLSTM model using embeddings obtained from the last layer of the BERT transformer model. The baseline is condition AGG, where we simply aggregate all the training data of the domains, and evaluate on the target domain in each column. In all domains except TC, AGG achieves better F1-scores than MAML and on average, AGG is better with over 2% F1-score.

Table 10 shows the result for the zero-shot cross-lingual adaptation. We make use of the NER datasets with three named entities: personal names, organisation and location. The language datasets we used are: CoNLL2002[17] (Dutch and Spanish), CoNLL2003[18] (English and German), Latvian,[19] Italian,[20] and French.[21] On average,

[14] Alexander Fritzler, Varvara Logacheva, and Maksim Kretov. "Few-Shot Classification in Named Entity Recognition Task". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. SAC '19. Limassol, Cyprus: Association for Computing Machinery, 2019, pp. 993–1000. ISBN: 9781450359337. DOI: 10.1145/3297280.3297378. URL: https://doi.org/10.1145/3297280.3297378; Yutai Hou et al. "Few-shot Slot Tagging with Collapsed Dependency Transfer and Label-enhanced Task-adaptive Projection Network". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 1381–1393. DOI: 10.18653/v1/2020.acl-main.128. URL: https://www.aclweb.org/anthology/2020.acl-main.128; Jiaxin Huang et al. *Few-Shot Named Entity Recognition: A Comprehensive Study*. arXiv:2012.14978. 2020.

[15] Jason Krone, Yi Zhang, and Mona Diab. "Learning to Classify Intents and Slot Labels Given a Handful of Examples". In: *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*. Online: Association for Computational Linguistics, July 2020, pp. 96–108. DOI: 10.18653/v1/2020.nlp4convai-1.12. URL: https://www.aclweb.org/anthology/2020.nlp4convai-1.12.

[16] https://catalog.ldc.upenn.edu/LDC2013T19

[17] Erik F. Tjong Kim Sang. "Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition". In: *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*. 2002. URL: https://www.aclweb.org/anthology/W02-2024.

[18] Erik F. Tjong Kim Sang and Fien De Meulder. "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition". In: *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*. 2003, pp. 142–147. URL: https://www.aclweb.org/anthology/W03-0419.

[19] Normunds Gruzitis et al. "Creation of a Balanced State-of-the-Art Multilayer Corpus for NLU". in: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: https://www.aclweb.org/anthology/L18-1714.

[20] B. Magnini et al. "I-CAB: the Italian Content Annotation Bank". In: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*. Genoa, Italy: European Language Resources Association (ELRA), May 2006. URL: http://www.lrec-conf.org/proceedings/lrec2006/pdf/518_pdf.pdf.

[21] Clemens Neudecker. "An Open Corpus for Named Entity Recognition in Historic Newspapers". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. Ed. by Nicoletta Calzolari (Conference Chair) et al. Portorož, Slovenia: European Language Resources Association (ELRA), May 2016, pp. 23–28. ISBN: 978-2-9517408-9-1.

Table 9: Zero-shot domain adaptation for NER on several domains of OntoNotes 5.0. F1-score (%) on the test set. The AGG and MAML methods are trained on sentences of all domains except the target domain (in each column) while the in-domain method (bottom row) is trained on sentences from the target domain only.

Method	BC	BN	MZ	NW	TC	WB	Average
AGG	**66.9**	**74.6**	**73.4**	**73.2**	64.7	**47.0**	**66.6**
MAML	65.4	72.7	71.8	62.8	**65.8**	46.1	64.1
In-domain	75.6	85.3	86.1	86.1	59.1	50.6	73.8

Table 10: Zero-shot cross-lingual transfer for NER on European languages. F1-score (%) on the test set. The AGG and MAML methods are trained on sentences of all languages except the target language (in each column) while the in-language method (bottom row) is trained on sentences from the target language only.

Method	DE	EN	ES	FR	IT	LV	NL	Average
AGG	**68.3**	**71.5**	68.4	59.7	**69.7**	**61.3**	75.8	67.8
MAML	67.7	70.8	**70.2**	**62.5**	69.4	60.6	**76.0**	**68.2**
In-language	78.0	89.6	84.0	68.1	76.8	78.8	86.8	80.3

the MAML approach is slightly better with a 0.4 F1-score improvement. Considering individual languages, AGG is better for German, English, Italian, and Latvian, however.

Overall, MAML did not perform significantly better than the baseline model (AGG). A possible future direction could be to explore other meta-learning algorithms like the Prototypical Networks for NER. In the meantime, we present a third approach that uses cloze-style prompts together with pre-trained LMs and performs well.

3.2.1.3. Leveraging pre-trained language models We now turn to our latest effort to provide a method for cost-effective SLU and DM implementations. As before, we focus on the NER task.

This approach is best motivated with an example. Let s = *"I will visit Munich next week"* be the sentence to label. As is typical, most words in s do not denote a named entity, only *"Munich"* does (`LOC`). For every detected entity, we would like to query a pre-trained LM to find out its type. Ideally, we would like to pass s to the pre-trained LM, then ask *"What is Munich?"* and receive *"a location"* as the answer. A cloze-style version of such a query to the pre-trained LM would be the following prompt:

> *"I would like to visit Munich next week. Munich is a [MASK]."*

The first part of the prompt is the sentence to label which serves as the context for the second part, a template of a predefined form, e.g., *"[TOKEN] is a [MASK]"*.[22]

[22] For non-masked LMs, [MASK] is the empty word.

The LM's prediction for the [MASK] is a probability distribution $P(w|s)$ over the tokens w in the vocabulary $\mathcal{V}$. Unfortunately, we found that directly looking up the probabilities for the named entity labels (e.g., the words *location, organisation*, etc.) and choosing the label with the highest probability did not perform well. However, by associating with each entity type a list of words representative of that type, we reach competitive performance (see Section 3.2.2). As an example, Table 11 shows representative words for three named entity classes, `Location`, `Person` and `Organisation`.

Table 11: Examples of the representative word lists for entity types `location` (LOC), `person` (PER), and `organisation` (ORG).

Entity Type	**Representative Words**
LOC	location, city, country, region, area, province, state, town, downtown
PER	person, man, woman, boy, girl, human, someone, kid
ORG	organisation, community, commission, department, association, company, union, team

More formally, let $\mathcal{L}$ be the set of all labels for our NER classification task. We provide a list of representative words $\mathcal{W}_l$ for each label l. Denoting the output of the LM by $P(.|s + T(w))$ where s is the original sentence, $T(w)$ is the prompt for token w, and $+$ stands for string concatenation, we easily assign label l_w to the token w by

$$l_w = \arg\max_l P(v \in \mathcal{W}_l | s + T(w)). \tag{10}$$

For the example above, ($s =$ *"I will visit Munich next week"; T("Munich")* $=$ *"Munich is a [MASK]."*), the top-5 predictions using the BERT-large model are: `city, success, democracy, capital, dream`. The largest probability (0.43) among all words is assigned to `city` which is among the representative words for label `LOC`. Thus, *Munich* is labelled as a location.

The outlined approach raises three design questions which we address below:

1. How to determine the tokens that refer to named entities?
2. What constitutes a good template for the second half of the prompt?
3. How to define lists of representative words for each entity type?

A fourth decision to make is the choice of a pre-trained LM. Our results of comparing four state-of-the-art pre-trained LMs are listed in Section 3.2.2.

Identifying named entities. Named entities (NEs) differ from other referents in that they are usually expressed as proper nouns, although some NE schemes also include entities such as numbers, date, and time expressions. Part-of-speech taggers can identify occurrences of proper nouns (or numerals, ordinals) with a high degree of accuracy. However, entity boundaries in case of multi-word expressions are usually not marked explicitly. We found treating consecutive proper nouns a single entity to be a reasonable heuristic.

Template selection. In order to gain insights into what makes a good template we have experimented with a number of variants (see Table 12). Overall, we found that

Table 12: A list of alternative templates used in the comparison experiments

ID	Template	Description
T_1	[TOKEN] is a [MASK].	The copula template is a straight-forward default that directly identifies the token with the mask.
T_2	[TOKEN] was a [MASK].	This template differs from T_1 only in the verb. We added it to study the influence of tense on the prediction quality. It lead to slightly worse results.
T_3	[TOKEN] would be a [MASK].	A variant of the above that uses a modal auxiliary. It performs similar to the past-tense copula.
T_4	[TOKEN] a [MASK].	We further experimented with stripping down a template to the minimum useful form. This template is ungrammatical, however, and does not perform well.
T_5	[TOKEN] [MASK].	This is the smallest template possible modulo word order. It is the worst performing template in our experiments, especially failing to predict the PER class.
T_6	[TOKEN] is an example of a [MASK].	These four templates are further variations of T_1 that replace the verb *to be* with longer constructions. They do not perform better than T_1, though.
T_7	[TOKEN] is an instance of a [MASK].	
T_8	[TOKEN] denotes a [MASK].	
T_9	[TOKEN] is well-known to be a [MASK].	
T_{10}	Many people consider [TOKEN] to be a [MASK].	In all previous templates, the token to label appears as the first word. Here, we test whether a longer left-hand side context is beneficial to the pre-trained LM prediction. In fact, our experiment shows a slight improvement over T_1 by 2%.
T_{11}	[TOKEN] is a common [MASK] known to many people.	With this template, we test the effect of extending the right-hand context. It does not produce the same performance gain as T_{10}, though.
T_{12}	There are many [MASK]s but [TOKEN] stands out nevertheless.	This template extends both the left-hand and the right-hand side context simultaneously, but also presents token and mask in a contrasting relation. The performance drops considerably, indicating that the LM has more difficulties associating the mask and the token with each other in this template.
T_{13}	A [MASK] like [TOKEN] is often mentioned in conversations.	Template T_{12} reverses the order of mask and token with respect to the previous templates. This template shows that the low performance of T_{12} is caused by this fact alone, as it is one of the best-performing templates in our experiments.
T_{14}	A [MASK] like [TOKEN].	Are the additional filler words responsible for the good performance of T_{13}, or is it the way the relation between mask and token are expressed using the word *like*? This reduced template suggests the latter, as it performs even slightly better than T_{13}.
T_{15}	This [MASK], [TOKEN], is worth discussing.	This template is similar in spirit to T_5 in that it tests whether proximity of mask and token are important, only with the order of the two reversed, and some context words added. It performs better than T_5 but not on par with most other templates.

most templates performed similarly, except where we intentionally tested the limits of the format. These experiments are detailed in Section 3.2.2.

Representative words. If some examples of the target domain are available, the representative word lists can be derived from predictions for this data. That is, the representative words for each entity type are selected from the most probable mask fillers given the respective prompts. Alternatively, in a zero-shot setting or where otherwise appropriate, a set of words can be provided by a domain expert. For the experiments in the next section, this is the strategy we followed. Of course, it is possible to combine both methods, with an expert choosing a suitable subset from words with the highest probability.

The method described thus far can be used for zero-shot classification as it does not require any training. In a few-shot setting however, we can improve the performance of the system by fine-tuning the pre-trained LM using the labelled data in the target domain.

A further performance gain can be made by combining our method with a standard supervised classifier in a simple two-fold ensemble. Based on a selection threshold p_h, we label the token according to the prediction of the prompt-based method if the probability of the predicted label is higher than the threshold p_h:

$$\max_l P(v \in \mathcal{W}_l | s + T(w)) > p_h, \tag{11}$$

otherwise we relay the output of the supervised classifier. The threshold p_h is a hyper-parameter that can be tuned on the training examples from the target domain. We call this setup the *hybrid approach*.

3.2.2. Experiments and evaluation

Comparing language models We study the role of the choice of pre-trained LM in our approach by comparing four pre-trained Transformer models: BERT,[23] RoBERTa,[24] GPT-2[25] and XLNET.[26] BERT and RoBERTa are trained to predict masked tokens that are randomly corrupted in the training sentence. GPT-2 is based on autoregressive language modeling, but it is less efficient predicting masked tokens. XLNET attempts to address the limitations of BERT for next-word prediction (i.e., autoregressive LM) while retaining good performance on Natural Language Understanding (NLU) tasks.

Table 13 compares the four LMs of the CoNLL03 dataset for the zero-shot setting as described above. We use the template *"[TOKEN]is a [MASK]"* for this experiment. We observe that BERT and RoBERTa have the best performance and GPT-2 is the worst when compared to the other models. This is expected, as BERT and RoBERTa

[23] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

[24] Y. Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv:1907.11692. July 2019.

[25] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf. 2019.

[26] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019, pp. 5753–5763. URL: https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.

Table 13: Performance comparison of four PLMs on the CoNLL03 dataset in zero-shot setting.

	BERT	**RoBERTa**	**GPT-2**	**XLNet**
LOC	69%	65%	42%	58%
PER	80%	73%	45%	57%
ORG	42%	43%	13%	34%
Avg.	60%	59%	36%	49%

Table 14: Comparing different templates T_1–T_{15} and their impact on the F1-score in the zero-shot setting on the CoNLL03 dataset. The results suggest that the performance of a template depends mainly on its naturalness.

Label	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
LOC	69%	60%	62%	52%	46%	66%	63%	60%	67%	69%	61%	57%	73%	70%	60%
PER	80%	72%	73%	65%	7%	81%	82%	71%	76%	82%	83%	15%	76%	81%	57%
ORG	42%	48%	51%	35%	35%	44%	39%	41%	47%	44%	41%	36%	48%	54%	34%
Avg.	60%	57%	57%	47%	31%	59%	57%	55%	60%	62%	59%	36%	62%	63%	49%

are trained to predict the masked token in the input text which is exactly how we formulated the NER task.

Choice of template A second variable in the setup is the choice of the template used in the prompt. We compare 15 different templates in the zero-shot setting (see Section 3.2.1.3). Table 14 presents the results of these experiments.

Domain adaptation We now assess the extent to which our prompt-based approach can improve the performance of available baseline methods for NER in a domain adaptation setting. Specifically, we are interested in a setting where knowledge of the source domain should be transferred to a target domain for which the number of available training samples is very limited.

We consider two baselines: in the *AGG* baseline, we merge the training data of the source and target domain and train an NER classifier on the resulting aggregated dataset. In the *Fine-tuning* baseline, we first train the model on the source domain and then fine-tune it on the training set of the target domain. Both of these approaches have been shown to reach competitive results in comparison with other state-of-the-art methods.[27] In both cases, a BERT-large cased pre-trained LM followed by a linear layer is used as the NER classifier.

In our first experiment, we also use the OntoNotes 5.0 dataset with the seven domains outlined in the previous section. However, we exclude pivot texts (bible) because they lack named entity annotations.

Following Wang et al.,[28] we take one domain of the dataset as the target domain

[27] Jing Li, Shuo Shang, and Ling Shao. "MetaNER: Named Entity Recognition with Meta-Learning". In: *Proceedings of The Web Conference 2020*. WWW '20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 429–440. ISBN: 9781450370233. DOI: 10.1145/3366423.3380127. URL: https://doi.org/10.1145/3366423.3380127.

[28] Jing Wang, Mayank Kulkarni, and Daniel Preotiuc-Pietro. "Multi-Domain Named Entity Recognition with Genre-Aware and Agnostic Inference". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 8476–8488. DOI: 10.18653/v1/2020.

and the other domains as the source domain. We randomly select $K = 100$ sample sentences from the target domain as our target training set. We also adopt a heterogeneous setup for selecting source and target labels. In particular, we choose `PERSON`, `ORG`, `PRODUCT` as source labels and `PERSON`, `ORG`, `PRODUCT`, `LOC`, `LANGUAGE`, `ORDINAL` as target labels. This discrepancy between target and source labels makes transfer learning more challenging.

Table 15 depicts the results of our experiments on the various OntoNotes 5.0 datasets averaged over five runs each. The table compares the performance of the baseline models with that of our hybrid approach. It also shows the results of the in-domain method, where we use all the training samples of the target domain for training the classifier. As is evident from this table, our prompt-based hybrid approach boosts the performance of all baseline models by a large margin.

Table 15: Domain adaptation for NER: F1-score of using the OntoNotes and i2b2 datasets. Combining our prompt-based method with the Fine-Tuning approach achieves the best performance. For all few-shot methods, we use K=100 samples of the target domain training set. In contrast, the in-domain method uses all available training samples and serves as a topline.

Method	**BC**	**BN**	**MZ**	**NW**	**TC**	**WB**	**i2b2**
AGG	46.3	47.9	46.9	52.7	51.7	43.8	53.5
AGG+Template	61.1	66.2	62.1	71.0	73.3	47.4	57.2
Fine-Tuning	66.7	71.2	69.3	74.1	65.2	**49.1**	62.3
Fine-Tuning+Template	**72.0**	**72.6**	**74.6**	**74.3**	**77.0**	**49.1**	**65.1**
In-domain	91.6	94.3	94.1	93.2	76.9	67.1	94.8

In our last experiment, we are interested in the impact of a greater discrepancy between source and target domain. We therefore take the OntoNotes 5.0 dataset as the source domain and the i2b2 2014 dataset[29] as the target domain, a BioNLP dataset commonly used for de-identification tasks. We use `PERSON`, `ORG`, `DATE`, `LOC` as source labels and `PERSON`, `ORG`, `DATE`, `LOC`, `PROFESSION` as target labels. The right-most column in Table 15 shows the results of our experiments. Again we observe the same pattern, i.e., combining our method with supervised baselines achieves the best performance.

4. Software library

4.1. Library for cost-effective learning of Speech-to-Text

The COMPRISE STT Weakly Supervised Learning library provides three main components which represent the cost-effective STT training approaches proposed in COMPRISE, namely:

- STT Error Detection driven Training.

acl-main.750. URL: https://www.aclweb.org/anthology/2020.acl-main.750.

[29] Amber Stubbs, Christopher Kotfila, and Özlem Uzuner. "Automated Systems for the De-Identification of Longitudinal Clinical Narratives". In: *J. of Biomedical Informatics* 58.S (Dec. 2015), S11–S19. ISSN: 1532-0464. DOI: 10.1016/j.jbi.2015.06.007. URL: https://doi.org/10.1016/j.jbi.2015.06.007.

- Weakly Supervised Training based on Dialogue States.
- Confusion Network based LM Training (CN2LM).

The first two components focus on obtaining reliable transcriptions of untranscribed speech data which can be used for training both STT AMs and LMs. They can be applied to any type of AM, although we choose the state-of-the-art Chain models in our examples. Readers interested in the high-level design, experimental evaluation and usage of these two components are directed to Deliverable D4.2.

The third component features training of statistical n-gram LMs and Recurrent Neural Network (RNN) LMs from alternate and uncertain STT hypotheses obtained on untranscribed speech data. This section provides prerequisites and typical usage-scenario of the CN2LM component only.

Source code and usage of these components is also available at:
`https://gitlab.inria.fr/comprise/speech-to-text-weakly-supervised-learning`

4.1.1. Prerequisites

- Numpy 1.19
- Pytorch 1.5.1 to train the CN2LM RNN version

4.1.2. Setup

If you plan to use the trained LMs for STT with Kaldi:

- Ensure that you have a working Kaldi installation.
- Modify the softlinks `steps` and `utils` in this directory to point to `egs/wsj/s5/steps/` and `egs/wsj/s5/utils/`, respectively, in your Kaldi installation.
- Modify the path of `KALDI_ROOT` and modify (or remove) the path to `kaldi_lm`, `SRILM` and sox tools in `path.sh`
- Modify `cmd.sh` if you are using a different execution queue for Kaldi.

4.1.3. Typical usage

CN2LM training will typically go through the following steps.

Step 1. Train seed STT models

- Supervised training data with reliable speech-transcript pairs are used to train the seed AM and LM. (Note that this step can be skipped if you already have pre-trained AM and LM).
- A sample Kaldi recipe to train the seed AM and LM on a subset of the Let's Go dataset is made available in the `egs/` directory.

Step 2. Prepare Confusion Networks

- The seed AM and LM are used to decode the unsupervised speech and dev sets into STT lattices. (Sample script in `egs/local/` if you are relying on sample recipe from Step 1.)
- Obtain STT confusion networks from the lattices decoded on the unsupervised speech and dev sets. The COMPRISE library assumes confusion networks are in Kaldi sausage format. Assuming your lattices are generated by Kaldi (as `lat.*.gz`), you can use our script to generate STT confusion networks as follows:

  ```
  bash local/err2unk/getSaus.sh lattice_dir graph_dir lm_wt
  ```

  'graph_dir' is the one used by the Kaldi decoder,

  'lm_wt' is the LM weight which gives the best dev set WER

  Note that STT confusion networks, aka sausages, are generated in the `<lattice_dir>/sau/` directory, referred as `saus_dir` in the next steps.

Step 3. Train CN2LM 3-gram LM

- A 3-gram LM can be trained on the combined supervised training speech transcripts and confusion networks obtained on the unsupervised speech as follows:

  ```
  python local/cn2lm/ngramlm/build_cn2lm_arpa.py asr_vocab_file
  sup_text unsup_saus_dir out_3glm_dir
  ```

  'asr_vocab_file' is the vocabulary following Kaldi's words.txt format

  'sup_text' is the supervised reference transcription in Kaldi format

  'unsup_saus_dir' is the unsupervised speech confusion networks directory generated in previous step

  'out_3glm_dir' is the output directory to store the 3-gram arpa LM

  Note that this CN2LM component has built-in features to train interpolated modified-KN smoothed 3-gram LMs only on reference transcriptions or only on confusion networks. It can also make use of error predictions on confusion networks to prune out the confusion network in non-error regions. Check `local/cn2lm/ngramlm/build_cn2lm_arpa.py` to use this pruning. Moreover, it has features to prune the maximum number of arcs seen in confusion bins. Check global `MAX_ARCS` in `local/cn2lm/ngramlm/data.py`.

Step 4. Train CN2LM RNN LM

- An RNN LM can be trained on the combined supervised training speech transcripts and confusion networks obtained on the unsupervised speech as follows:

  ```
  python local/cn2lm/rnnlm/train_cn2lm_rnn.py asr_vocab_file
  sup_text unsup_saus_dir dev_saus_dir dev_text out_rnnlm_dir
  ```

  'asr_vocab_file' is the vocabulary following Kaldi's words.txt format

  'sup_text' is the supervised reference transcription in Kaldi format

'unsup_saus_dir' is the unsupervised speech confusion networks directory generated in previous step

'dev_saus_dir' is the dev set confusion networks directory generated in previous step

'out_rnnlm_dir' is the output directory to store the RNN LM model in Pytorch's pth format

Note that this CN2LM component has built in features to train LSTM or GRU RNN LM, sharing of input-output word embedding layers and support for different pooling schemes over confusion bin arcs. Moreover, it has features to prune the maximum number of arcs seen in confusion bins. Check globals in `local/cn2lm/rnnlm/models.py and local/cn2lm/rnnlm/data.py`.

- Support is provided to convert CN2LM GRU RNN LM to Kaldi RNN LM format as follows:

```
bash local/cn2lm/rnnlm/kaldi_support/pytorch_rnnlm_to_kaldi.sh
asr_vocab_file kaldi_gru_lm_template_file pytorch_model
out_kaldi_model_dir
```

'asr_vocab_file' is the vocabulary following Kaldi's words.txt format

'kaldi_gru_lm_template_file' is Kaldi nnet3 format template file. A template for a single layer RNN LM with shared input-output embeddings and Pytorch GRU cell is provided in `local/cn2lm/rnnlm/kaldi_support/`

'pytorch_model' is the Pytorch format RNN LM trained in the previous step

'out_kaldi_model_dir' will store Kaldi compatible RNN LM files

Note that this step currently supports only single layer RNN LM with shared input-output embeddings and Pytorch GRU cell. Support for more layers and LSTM can be easily added if a suitable `kaldi_gru_lm_template_file` is created.

4.2. Library for cost-effective learning of text processing

4.2.1. Prerequisites

After retrieving a copy of the library, and before the software can be run, four dependencies need to be installed:

- `PyTorch` - A popular machine learning library for training models.
- `Transformers` - A library that provides pre-trained language models.
- `spaCy` - A library for Part-of-Speech tagging.
- `Seqeval` - A library for measuring the performance.

Detailed installation instructions for `PyTorch` can be found under `https://pytorch.org/`. Similarly, the website `https://spacy.io/usage` provides detailed installation instructions for `spaCy`. For the other two dependencies, we recommend to install them through `pip`:

```
pip install transformers seqeval
```

4.2.2. Configuration

The software is invoked with a configuration file that contains the following information:

- `Template` - *"[TOKEN] is a [MASK]"* (without quotes) has been found to work well in all cases.
- `List of representative words` - For each class of labels the user has to provide a list of support words that associated with the class.
- `Number of shots in target domain` - In the case of few-shot fine-tuning.
- `Threshold` - A parameter used in the prediction phase (see above).

The template, the list of words, and the threshold can be considered as hyper-parameters and may be chosen either off-hand or, ideally, so as to maximise the classifier performance on a validation set.

These configuration options are specified as members of a JSON[30] object, using the keys `"template"`, `"labels"`, `"threshold"`, `"shots"`. The value of `"template"` is a string that must contain the substrings `[TOKEN]` and `[MASK]` - these mark the position in the template that will be filled by the system during prediction. The value of the key `"labels"` is another JSON object, in which the keys are the Named Entity labels to be used, and the value for each such key is a list of strings representing the support words, chosen by the user. If few-shot learning is employed, the value of the key `"shots"` is a positive integer, specifying the number of samples that are randomly sampled from the training data in order to fine-tune the BERT model. Finally, the value of key `"threshold"` must be a floating point value between 0.0 and 1.0.

Figure 5 provides an example of such a configuration file.

4.2.3. Data format

For fine-tuning and for testing the classifier, the text should be in BIO format. This format requires that the sentences to classify are stored in a plain text file, with each

[30] `https://www.json.org`

```
{
 "template": "[TOKEN] is a [MASK]",
 "labels": {
   "LOC": ["city", "country", "region", "area"],
   "PER": ["man", "woman", "child"],
   "ORG": ["organisation", "company", "club"],
 },
 "shots": 100,
 "threshold": 0.8,
}
```

Figure 5: An example configuration file for the template-based approach.

word on a separate line. Sentence boundaries are marked by an empty line. Each word's label should come after the word in the same line, separated by a TAB character ("\t"). Here is an example of this format:

```
Arrived O
at      O
Tyndrum B-LOC
at      O
about   B-TIME
two     I-TIME
o'clock I-TIME
.       O
```

Named entities that consist of more than one word are labelled such that the first label has the prefix `B-` and the remaining words all have the prefix `I-` (see *"about two o'clock"* in the example above). Single-word entities are labelled with the prefix `B-` (see *"Tyndrum"*). Words that do not constitute any Named Entity are labelled with the label `O`.

The text to be labelled should also be formatted in the same way, except that no labels need to be given. The text would be provided as a plain text file with only a single word on each line, and empty lines marking sentence boundaries.

4.2.4. Running the code

The classifier can be run out-of-the-box without any prior training (zero-shot setting). However, to achieve better performance, we recommend a fine-tuning step with a small number of labelled in-domain samples. In any event, the procedure to classify an input text using the template method consists of two steps.

In the first step, we train a classifier on top of a pre-trained language model using supervised learning:

```
python run_ner_write_probs.py --data_dir <datadir> \
                      --model_name_or_path <model-path>
```

Here, `data_dir` provides the path to the NER dataset. It should contain a file with training data, called `train.txt`, and a file with test data called `test.txt`. The parameter `model_name_or_path` gives the name or path to the pre-trained language model. The predictions of this model are written to `<datadir>` and are subsequently used by the hybrid model.

We execute our hybrid model as a second step via the following command:

```
python template.py --data_dir <data-dir> \
                   --config <path_to_config_file> \
                   [--output_file <outfile>]
```

It takes the predictions of the supervised classifier as well as train and test data as inputs and optionally writes the predicted tags in a separate file. Again, `<data-dir>` refers to a directory containing training and test data of the NER task as well as the

predictions of the supervised model (This corresponds to `output_folder` in the first step). The parameter `--config` specifies the path to the configuration file (see Section 4.2.2). The optional parameter `<outfile>` specifies the file to which the predictions are written. If it is omitted, and the input file contains gold-standard annotations, then only the classification performance is reported.

5. Summary and outlook

The COMPRISE library for weakly supervised learning provides software tools for weakly supervised training of STT and NER in text.

The tools for training STT models retain the two main approaches proposed in the initial library, namely training guided by STT error predictions (Err2Unk) and weak supervision with dialogue states. A new STT LM training component is provided, which can be used independently of the components from the initial library. The new Confusion Network based LM (CN2LM) component supports training of both n-gram and RNN LMs.

The evaluation of semi-supervised 3-gram and RNN LMs trained using CN2LM showed significant reduction in perplexity on the Verbmobil dataset, containing human conversations. Interestingly, the lowest WER with 3-gram LMs was obtained with semi-supervised AM and LM based on the Err2Unk approach. However, CN2LM based 3-gram LM consistently performed better than Bestpath semi-supervised LMs. The reduced perplexity and consistent improvements over Bestpath semi-supervision make CN2LM a potential approach to be tried alongside, or in combination with, Err2Unk semi-supervision.

The evaluation of CN2LM on the Let's Go dataset showed that it performs equally well as Bestpath and Err2Unk semi-supervised LMs. It can be noted that the improvements in this setup largely come from the semi-supervised AM, in particular the Err2Unk semi-supervised AM. Moreover, the Err2Unk approach with lowest WER has a 1% higher WER than the oracle models. This makes us conclude that setups similar to the Let's Go system, which mainly involve a limited variety short length queries, can be handled by Err2Unk semi-supervised AM and LM. However, as demonstrated in Deliverable D4.2, dialogue state based weak supervision can give significant performance gains in such a setup if utterance level dialogue state labels are available.

Owing to the less than ideal performance of our initial weakly supervised learning approach for text processing, we have looked at alternative options towards the goal of cost-effective SLU and DM. In this document, we have thus reported on two independent efforts: Meta-Learning and a cloze-style template-based approach for NER.

Meta-Learning has received quite a lot of attention in the computer vision community but is still a fairly new technique for NLP. We applied Model-Agnostic Meta-learning to the NER task in a zero-shot setting, i.e., where no annotations for the target domain are available. Unfortunately though, we found that the approach performs worse than simply aggregating the source domains into a common training set for a standard supervised learning approach.

This motivated yet another approach. As recent research has pointed at pre-trained language models as general knowledge representations, we devised a template-based approach to zero-shot and few-shot learning for NER. The key idea is that

with the help of a list of representative words for each class label, ordinary cloze-style queries can be submitted to a BERT model.

Perhaps the most attractive property of the approach is that it scales seamlessly down to even a zero-shot setting, requiring no training at all. However, as we demonstrate in our experiments, our model benefits from even as low as 100 labelled training examples. Compared to the amount of data typically required for supervised learning, this is still a very modest requirement, directly translating to enormous cost savings for practitioners.

Our software is made publicly available under an Open Source license. The two parts of the final weakly supervised learning library can be accessed here:

- Speech-to-text:
  https://gitlab.inria.fr/comprise/speech-to-text-weakly-supervised-learning
- Text processing:
  https://gitlab.inria.fr/comprise/spoken-language-understanding-weakly-supervised-learning