FINET: Context-Aware Fine-Grained Named Entity Typing

Luciano Del Corro* **Abdalghani Abujabal*** **Rainer Gemulla†** **Gerhard Weikum***
* Max-Planck-Institut für Informatik, Saarbrücken, Germany
{delcorro, abujabal, weikum}@mpi-inf.mpg.de
† Universität Mannheim, Mannheim, Germany
rgemulla@uni-mannheim.de

Abstract

We propose FINET, a system for detecting the types of named entities in short inputs—such as sentences or tweets—with respect to WordNet's super fine-grained type system. FINET generates candidate types using a sequence of multiple extractors, ranging from explicitly mentioned types to implicit types, and subsequently selects the most appropriate using ideas from word-sense disambiguation. FINET combats data scarcity and noise from existing systems: It does not rely on supervision in its extractors and generates training data for type selection from WordNet and other resources. FINET supports the most fine-grained type system so far, including types with no annotated training data. Our experiments indicate that FINET outperforms state-of-the-art methods in terms of recall, precision, and granularity of extracted types.

1 Introduction

Named entity typing (NET) is the task of detecting the type(s) of a named entity in context. For instance, given "John plays guitar on the stage", our goal is to infer that "John" is a *guitarist* or a *musician* and a *person*. We propose FINET, a system for detecting the types of named entities in short inputs—such as sentences or tweets—with respect to WordNet's super fine-grained type system (16k types of organizations, persons and locations).

Named entity typing is a fundamental building block for many natural-language processing tasks. NET is at the heart of information extraction methods for finding types for entities in a knowledge base[1] (KB) (Mitchell et al., 2015). Likewise, NET aids named entity disambiguation by reducing the candidate space for a given entity mention. Entity types are an important resource for entity-based retrieval or aggregation tasks, such as semantic search (Hoffart et al., 2014) or question answering (Yahya et al., 2013). Finally, type information helps to increase the semantic content of syntactic patterns (Nakashole et al., 2012) or in open information extraction (Lin et al., 2012).

The extraction of explicit types has been studied in the literature, most prominently in the context of taxonomy induction (Snow et al., 2006). Explicit types occur, for example, in phrases such as "Steinmeier, the German Foreign Minister, [...]" or "Foreign Minister Steinmeier." These explicit types are often extracted via patterns, such as the well-known Hearst patterns (Hearst, 1992), and subsequently integrated into a taxonomy. Pattern-based methods often have high precision but low recall: Types are usually mentioned when a named entity is introduced or expected to be unknown to readers, but often are not explicitly stated. The NET problem differs from taxonomy induction in that (1) the type system is prespecified, (2) types are disambiguated, and (3) types are associated with each occurrence of named entity in context.

Our FINET system makes use of explicit type extractions whenever possible. But even when types are not explicitly mentioned, sentences may give clues to the correct type. These clues range from almost explicit to highly implicit. For example, in "John plays soccer", the type *soccer player* is almost explicit. The sentence "Pavano never even made it to the mound," however, only implicitly indicates that "Pavano" is a *baseball player*. A

[1] In this paper, we refer to WordNet as a type system and to a collection of entities and their types as a KB.

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 868–878,
Lisbon, Portugal, 17-21 September 2015.

key challenge in NET is to extract such implicit, context-aware types to improve recall.

One way to extract implicit types is by training a supervised extractor on labeled data, in which each entity is annotated with appropriate types. The key problem of this approach is that training data is scarce; this scarcity is amplified for fine-grained type systems. To address this problem, many existing systems generate training data by exploiting KBs as a resource (Yosef et al., 2012). A popular approach is to train an extractor on a corpus of sentences (e.g., on Wikipedia), in which each named entity is associated with all its types in a KB. The key problem with such an approach is that the so-obtained type information is oblivious to the context in which the entity was mentioned. For example, in the sentences "Klitschko is known for his powerful punches" and "Klitschko is the Mayor of Kiew," "Klitschko" will be associated with all its types, e.g., *boxer*, *politician* and *mayor*. As a consequence, the labels in the training data can be misleading and negatively affect the extractors. Moreover, such learned extractors are often biased towards prominent types but perform poorly on infrequent types, and they are generally problematic when types are correlated (e.g., most *presidents* are also *graduates* and *authors*).

FINET addresses the above problems by first generating a set of type candidates using multiple extractors and then selecting the most appropriate type(s). To generate candidates, we make use of a sequence of extractors that range from explicit to highly implicit. Implicit extractors are only used when more explicit extractors fail to produce a good type. Our extractors are based on patterns, mention text, and verbal phrases. To additionally extract highly implicit types, we use word vectors (Mikolov et al., 2013) trained on a large unlabeled corpus to determine the types of *similar* entities that appear in *similar* contexts. This extractor is comparable to KB methods discussed above, but is unsupervised, and takes as candidates the types frequent within the related entities and contexts.

After type candidates have been generated, the final step of FINET selects the types that best fit the context. In this step, we leverage previous work on word sense disambiguation (WSD) and resources such as WordNet glosses, and, if available, manually annotated training data.

FINET leverages ideas from existing systems and extends them by (1) handling short inputs (2) supporting a very fine-grained type hierarchy, and (3) producing types that focus on the entity context. Existing systems are unable to extract more than a couple of hundred types. Hyena (Yosef et al., 2012), the system with the most fine-grained type system so far, focuses only on a subset of 505 types from WordNet. Hyena lacks important types such as *president* or *businessman*, and includes *soccer player* but not *tennis player*. Instead of restricting types, FINET operates on the the entire WordNet hierarchy with more than 16k types for persons, organizations, and locations.

We evaluated FINET on a number of real-world datasets. Our results indicate that FINET significantly outperforms previous methods.

Extractor	Stopping Condition
Pattern-based (final)	Always stop
Pattern-based (non-final)	KB-lookup
Mention-based	KB-lookup
Verb-based	KB-lookup
Corpus-based	$\geqslant 50\%$ of score in $\leqslant 10$ types

Table 1: Extractors and their stopping conditions

2 Candidate Generation

In this phase, we collect candidate types for each entity. We first preprocess the input (Sec. 2.1) and then apply a (i) pattern-based extractor (Sec. 2.2), (ii) a mention-based extractor (Sec. 2.4), (iii) a verb-based extractor (Sec. 2.5), and (iv) a corpus-based extractor (Sec. 2.6). The extractors are ordered by decreasing degree of explicitness.

Each extractor has a *stopping condition*, which we check whenever the extractor produced at least one type. When the stopping condition is met, we directly proceed to the type selection phase. The reasoning behind this approach is to bias FINET towards the most explicit types. When the condition is not met, we enrich the set of candidate types of the extractor with their hypernyms; we expect types to be overly specific and want to allow the selection phase to be able to select a more general type. We then also run subsequent extractors. Tab. 1 displays a summary of the extractors and stopping conditions.

All so-found type candidates are passed to the candidate selection phase (Sec. 3).

2.1 Preprocessing

Preprocessing consists of 5 steps: (i) dependency parsing (Socher et al., 2013); (ii) co-reference (Recasens et al., 2013); (iii) named entity recogni-

tion (NER) (Finkel et al., 2005) with the detection of coarse-grained types (i.e., *person*, *organization*, *location*); (iv) clause identification (Del Corro and Gemulla, 2013); (v) word and multi-word expression recognition (Del Corro et al., 2014).

FINET restricts its candidates to the hyponmys of the coarse-grained (CG) type of the NER system. Named entities with the same CG type in a coordinating relation (e.g., "Messi and Ronaldo are soccer players") and identical mentions share the candidate set; the latter is reasonable in short input.

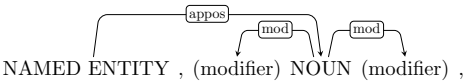
FINET extractors operate either on the sentence or the clause level. A clause is a part of a sentence that expresses a statement and is thus a suitable unit for automatic text processing (Del Corro and Gemulla, 2013). Finally, we identify multi-word explicit type mentions such as *Prime Minister* or *Secretary of Housing and Urban Development* (Del Corro et al., 2014).

2.2 Pattern-based extractor

Our pattern-based extractor targets explicit type mentions. These are commonly used to introduce entities when they first appear ("US President Barack Obama") or when their mention does not refer to the most prominent entity ("Barack Obama, father of the US President"). Following previous work (Hearst, 1992), we use a set of patterns to look for expressions that may refer named entity types. We refer to those expressions as *lexical types* (e.g., "father"). Once lexical types have been identified, we collect as candidate types the WordNet synsets to which they refer (e.g., ⟨*father*-1⟩, ..., ⟨*father*-8⟩, the eight senses of "father").

Our extractor makes use of both syntactic patterns, which operate on the dependency parse, and regular expression patterns, which operate on the text. Syntactic patterns are preferable in that they do not rely on continuous chunks of text and can skip non-relevant information. However, mistakes in the dependency parse may lower recall. To cope with this, we additionally include regular expressions for some syntactic patterns. Fig. 1 shows an example of a syntactic pattern and a related regular-expression pattern. Both produce lexical type "president" from "Barack Obama, president of the US," but only the syntactic pattern applies to "Barack Obama, the current US president."

Tab. 2 gives an overview of our patterns. Most of them also have a symmetric version (e.g., "The president, Barack Obama" and "Barack Obama,



(a) A syntactic pattern

NAMED_ENTITY , NOUN

(b) A regular expression pattern

Figure 1: Patterns capturing appositions

Pattern	Example
Final patterns	
Hearst I	{Presidents} such as [Obama] (and) [Bush]
Hearst II	{Presidents} like [Obama] (and) [Bush]
Hearst III	Obama (and) other {presidents}
Hearst IV	{Presidents} including [Obama] (and) [Bush]
Apposition	[Obama], (the) {president}
Copular	[Obama] is (the) {president}
Noun modifier	{President} [Barack Obama]
Among	[Joe Biden] among (other) {vice presidents}
Enough	[Messi] is enough (of) a {player}
As	[Messi] as {player}
Non-final patterns	
Location	{City} of [London]
Poss. + transf.	[Shakespeare]'s {productions}
by-prep + transf.	{productions} by [Shakespeare]

Table 2: Patterns for explicit type extraction

the president"), which is not displayed. We divide our patterns into final and non-final. Final patterns generally have high precision and extract the lexical type exactly as it occurs. When a final pattern produces a lexical type, we add the corresponding types to the candidate set and go directly to the type selection phase, i.e., we do not consider any other extractor. For non-final patterns, however, we expect erroneous extractions and proceed differently; we perform a KB lookup for all lexical types. The KB lookup (described in the next section) both prunes and expands the candidate set using a KB, and acts as a stopping condition. (FINET can also be run without using KB lookups.)

We treat a pattern as non-final if it may or may not denote a lexical type (e.g. "the president of Argentina" vs. "the city of Buenos Aires") or if a verb-noun transformation is required to obtain the lexical type (e.g. "Shakespeare's productions" to "producer"). To perform the transformations, we use WordNet's *derivationally related forms*, which connect semantically related morphological variations of verbs and nouns. For instance, the noun "production" is connected to the verb "produce," which in turn is connected to the noun "producer". These variations can be exploited to extract explicit types. We treat such transforma-

tions as non-final because we may make mistakes. Moreover, WordNet is highly incomplete in terms of derivational forms. For instance, we may not be able to reach all the possible senses for "producer" from "production," and from "works" in "works by Schumann", we cannot reach "musician" or "artist," but we reach "worker" and there is no synset of "worker" that specifically denotes an artist.

2.3 Exploiting a knowledge base

Most of our extractors (optionally) leverage a knowledge base to (1) prune candidate types, (2) find additional candidates, and (3) decide whether to consider subsequent extractors. To do so, we extract from a KB a repository of (entity mention, type)-pairs.[2] We use the KB conservatively, i.e., we consider KB evidence only if the extracted types match the ones in the KB.

The KB is leveraged via a *KB lookup*. Each KB lookup takes as input an entity mention e and a set T of candidate types found by the extractor (lexical or disambiguated). We first replace each lexical type in T by the set of the corresponding types from WordNet. Afterwards, for each type $t \in T$, we check whether there is one or more matching types (e, t_{KB}) in the KB for the given mention. Type t_{KB} matches t if it is either identical, a hypernym, or a hyponym of t. For each match, the KB lookup outputs t. If t_{KB} is a hyponym of t, we additionally output t_{KB}, that is, a type more specific than the one found by the extractor. We leave the decision of whether t or t_{KB} is a more suitable type to the type selection phase. For example, for e = "Messi" and $t = \langle$*player*-1$\rangle$, we output $\langle$*player*-1$\rangle$ and $\langle$*soccer_player*-1$\rangle$ (a hyponym).

The KB lookup is *successful* if it outputs at least one type. Then we add the resulting types to the candidate set and go directly to the type selection phase. A KB lookup *fails* if no matching type was found. We then proceed differently: if a KB lookup fails, we add the complete set T to the candidate set and continue to the next extractor.

As mentioned above, FINET can also be run without any KB lookups; the corresponding extractors then do not have a stopping condition. In Sec. 4, we experimented with both variants and found that KB lookups generally helped.

[2]We used Yago2 (Hoffart et al., 2013). Our repository contained roughly 9M entity mentions and 20M pairs.

2.4 Mention-based extractor

Our second extractor aims to extract type candidates from the entity mention. This is particularly effective for organizations, which often contain the lexical type in their name (e.g., "Johnson & Wales University" or "Republican House").

Given an entity mention, we check if any of the words or expressions in the name corresponds to a lexical type in WordNet. If so, we consider the corresponding types as potential candidates. For instance, for "Imperial College London", we extract "college" and obtain types $\langle$*college*-1$\rangle$ and $\langle$*college*-2$\rangle$ (both matching the CG type) from WordNet. We then perform a KB lookup.

We extend the above procedure for entities tagged as *location*. Since the set of (named-entity) locations is quite static and known, we assume that the KB contains all locations and their possible types (our experiments strengthened this assumption). If a mention of a location (e.g., "Berlin") occurs in the repository, we add all the corresponding types from the repository to the candidate set (e.g., $\langle$*city*-1$\rangle$) and move to the type selection phase.

2.5 Verb-based extractor

Verbs have been widely exploited to determine the types or roles of its arguments: A verb sense imposes a restriction on the type of its arguments (Quirk et al., 1985; Levin, 1993; Hanks, 1996; Baker et al., 1998; Palmer et al., 2005; Kipper et al., 2008). For instance, from "Ted Kennedy was elected to Congress," we infer that "Ted Kennedy" is a *person* who can be elected. Corresponding types include $\langle$*representative*-1$\rangle$, $\langle$*representative*-2$\rangle$, or $\langle$*politician*-1$\rangle$. Our verb-based extractor leverages this insight to extract types. The extractor operates at the clause level.

A simple way to infer lexical types for entities acting as subjects or objects of a clause is *nominalization*, i.e., the transformation of the verb into *deverbal nouns* (e.g., "play" into "player"). To exploit it, we apply a set of morphological transformations to the verb (Quirk et al., 1985), which depend on the grammatical function of the entity, i.e., subject or object. If the entity mention acts as a subject, we try adding the suffixes "-er,", "-or," and "-ant" to the verb's lemma. If the mention acts as an object, we use suffixes "-ee" and "-ed". To obtain candidate types, we again use derivationally related forms (DER). We consider as potential candidates all types referred to by one of the deverbal nouns and connected to a sense of the

verb via DER. For instance, given "Messi plays in Barcelona," we collect for "Messi" all the senses of "player" that are connected to some sense of "play" (⟨*player*-1⟩, ⟨*musician*-1⟩ and ⟨*actor*-1⟩).

We also explore WordNet in a way that is not restricted to morphological variations of the verb. For instance, in "John committed a crime," "commit" is a synonym of "perpetrate," which in turn can be varied to "perpetrator". We consider the morphological variations of all synonyms of the verb. Moreover, if the named entity is the subject of the clause, and if the clause contains a direct object, we form a new lexical type by adding the direct object as a noun modifier of the deverbal noun (e.g., From "Messi plays soccer", we form "soccer player"). If it exists in WordNet, we consider the respective types as potential candidates as well.

A more indirect way of exploiting the verb-type semantic concordance is via a corpus of frequent (verb, type)-pairs, where the type refers to possible types of the verb's subject or object. As stated above, the set of argument types compatible with a verb is limited. For instance, "treat" is usually followed by ⟨*condition*-1⟩, ⟨*disease*-1⟩, or ⟨*patient*-1⟩. FINET, uses the corpora of Flati and Navigli (2013) and Del Corro et al. (2014). Given a verb and an entity, we search for frequent candidate types (depending on whether the entity acts as a subject or object). For example, from "Messi was treated in the hospital," we obtain ⟨*patient*-1⟩.

Once potential candidates have been collected, we perform a KB lookup to decide how to proceed.

2.6 Corpus-based extractor

Our final extractor leverages a large unlabeled corpus to find entities that co-occur in similar contexts. It is based on the distributional hypothesis (Sahlgren, 2008): similar entities tend to occur in similar contexts. For example, "Messi" and "Cristiano Ronaldo" may both be mentioned in the context of soccer. Thus entity mentions similar to "Messi" in a sport context are likely to include soccer players. Our extractor is related to semi-supervised KB methods in that it propagates types of named entity mentions that may appear in a similar context. It differs in that it is unsupervised, does not require manually or automatically generated training data, and in the way context is modeled and candidates are generated.

Our corpus-based extractor makes use of word vectors (Rumelhart et al., 1988) trained on a large unlabeled corpus. A word vector is a semantic representation of a phrase and represents the semantic context in which the phrase occurs. Phrases that are semantically related, and thus appear in similar contexts, are close to each other in the word vector space. For instance, if "Messi" and "Cristiano Ronaldo" tend to co-occur with a similar sets of words, their word vectors are close. We also may expect "Arnold Schwarzenegger" to be close to both actors and politicians, since it occurs in both contexts. In our work, we use word2vec (Mikolov et al., 2013), which provides a model trained on Google News to predict related words or phrases for a *query* specified as a set of phrases. Given an integer k, word2vec outputs the set of k phrases that are most similar to the query.

Our corpus-based extractor uses (1) the input sentence to construct a set of relevant queries and (2) the word2vec results and a KB. To construct a query for a given entity mention, we focus on the part of the sentence directly related to the entity. This relevant part consists of the clause in which the entity occurs and the subordinate clauses that do not contain another entity. Since word2vec is most effective when queries are short, we construct a set of small queries, each consisting of the named entity mention and some context information. We construct a query for each noun phrase (of length at most 2) and for each other entity mention. If the named entity occurs as subject or object, we also take the corresponding verb and the head of the object or subject. For example, the queries for "Maradona expects to win in South Africa" are {"Maradona", "South Africa"} and {"Maradona", "expect", "win"}.

For each query, we retrieve the 100 most related phrases with their similarity score and union the results. We filter them using our KB and retain only those phrases that correspond to entity mentions (with the correct CG types). We then enrich each mention by the set of their possible types from the KB. Here we exclude widespread but irrelevant implicit types such as ⟨*male*-1⟩, ⟨*female*-1⟩, ⟨*adult*-1⟩, ⟨*commoner*-1⟩, ⟨*friend*-1⟩, or ⟨*alumnus*-1⟩. We also include the types corresponding to the entity mention (with score 1). If there is sufficient evidence that some of the so-obtained types are most prominent, we take these types as candidates. In our example, query {"Maradona" "South Africa"}, all of the top-15 persons (e.g., "Diego Maradona", "Carlos Alberto Parreira", "Dunga", "Beckenbauer") share type ⟨*coach*-1⟩; a strong indication that Maradona may

also be of type ⟨*coach*-1⟩. To select prominent types we traverse the results until we collect 50% of the total score. We take all so-collected types as candidates. If no more than 10 different types were added this way, we directly go to the type selection phase.

3 Type Selection

The type selection phase selects the most appropriate type from the set of candidates of a given named entity. We use techniques from WSD, but adapt them to our setting. WSD aims to disambiguate a word or phrase (e.g., a noun) with respect to a type system as WordNet; e.g., from "player" to ⟨*player*-1⟩. The main difference between WSD and our type selection is that our goal is to decide between a set of types for an entity mention; e.g., from "Messi" to ⟨*soccer_player*-1⟩. Our type selection step can be used as-is; it is not trained on any domain- or corpus-specific data.

3.1 Obtaining context

All WSD systems take a set of candidate types and contextual information as input. The key challenge lies in the construction of candidate types (Sec. 2) as well as context. For each entity, we consider *entity-oblivious context* (from the input sentence) as well as *entity-specific context* (using lexical expansions).

We take all words in the sentence as entity-oblivious context. To construct entity-specific context, we make use of lexical expansions, which have been successfully applied in WSD (Miller et al., 2012). Its goal is to enrich contextual information to boost disambiguation. In our case, it also helps to differentiate between multiple entities in a sentence. We build the entity-specific context using word vectors. As in the corpus-based extractor, we construct a set of queries for the entity but in this case we take as context all so-obtained words that do *not* correspond to a named entity. For instance, the entity-specific context for the entity mention "Maradona" for query "Maradona South_Africa" is: "coach", "cup", "striker", "midfielder", and "captain". The full context for "Maradona" in "Maradona expects to win in South Africa" additionally includes the entity-oblivious context "expects", "win", "South Africa".

3.2 Selecting types

WSD systems fall into two classes: unsupervised, which rely on background knowledge such as WordNet (Ponzetto and Navigli, 2010), and supervised, which require training data (Zhong and Ng, 2010). Here we take a combination, i.e., we leverage WordNet and manually annotated data.

We train a Naive Bayes classifier to select the most appropriate type given its context. We represent context by a bag of lemmatized words. This allows us to automatically generate training data from WordNet (and use manually labeled data). Since WordNet provides information for each of the relevant types, this approach combats the data sparsity that arises with supervised systems. The context for each individual WordNet type consists of all words appearing in the type's gloss and the glosses of its neighbors (Banerjee and Pedersen, 2003). We also include for each type the neighbors from Ponzetto and Navigli (2010) and the corresponding verbs from Del Corro and Gemulla (2013). Finally, we add all words in sentences containing the type in SemCor[3] (Landes et al., 1998) and Ontonotes 5.0 (Hovy et al., 2006).

We trained one classifier per CG type. To train the classifier, we create a single training point for each corresponding WordNet type and use the type's context as features. To map the CG types from our NER system to WordNet, we considered as persons all descendants of ⟨*person*-1⟩, ⟨*imaginary being*-1⟩, ⟨*characterization*-3⟩, and ⟨*operator*-2⟩ (10584 in total); as locations all descendants of ⟨*location*-1⟩, ⟨*way*-1⟩, and ⟨*landmass*-1⟩ (3681 in total); and as organizations all descendants of ⟨*organization*-1⟩ and ⟨*social group*-1⟩ (1968 in total). This approach of handling CG types suffers to some extent from WordNet's incompleteness, esp. with respect to persons and organizations. For instance, phrase "sponsored by Coca-Cola" implies that "Coca-Cola" is a "sponsor," but in WordNet, only persons can be sponsors. Nevertheless, this approach worked well in our experiments.

4 Experiments

We conducted an experimental study on multiple real-word datasets to compare FINET with two state-of-the-art approaches. FINET is used as-is; it does not require training or tuning for any specific dataset. All datasets, detected types, labels, and our source code are publicly available.[4]

[3] http://web.eecs.umich.edu/~mihalcea/downloads.html

[4] http://dws.informatik.uni-mannheim.de/en/resources/software/finet/

4.1 Experimental Setup

Methods. *Hyena (Yosef et al., 2012)* is a representative supervised method that uses a hierarchical classifier. Its features include the words in the named entity mention, in sentence and paragraph, and POS tags. It performs basic co-reference resolution and marks entity mentions connected to a type in the KB using a binary feature. Similar to Ling and Weld (2012), Hyena is trained on Wikipedia entities, each being annotated with its corresponding WordNet types from YAGO. Hyena's type system is restricted to 505 WordNet types with top categories ⟨*artifact*-1⟩, ⟨*event*-1⟩, ⟨*person*-1⟩, ⟨*location*-1⟩, and ⟨*organization*-1⟩. Hyena outperformed a number of previous systems (Fleischman and Hovy, 2002; Rahman and Ng, 2010; Ling and Weld, 2012). We used Hyena via its web service (Yosef et al., 2013).

Pearl (Nakashole et al., 2013) is a semi-supervised system that leverages a repository of 300k relational patterns (Nakashole et al., 2012). Subjects and objects of each pattern carry type information. Pearl types named entity mentions by the most likely type according to its pattern database. Pearl's type system is based on around 200 "interesting" WordNet types. We ran Pearl in its *hard* setting, which performed best.

FINET. We ran FINET in two configurations: (1) with KB lookup, (2) without the KB lookup. This allows us to estimate the extent to which referring to a KB helps. Note that the corpus-based extractor makes use of the KB in both settings.

Datasets. We used three different datasets representing real-world use cases. We created two datasets, New York Times and Twitter, and sampled a subset of the CoNLL data, which provides gold annotations for CG types. We did not consider datasets such as FIGER (Ling and Weld, 2012) or BBN (Weischedel and Brunstein, 2005) because they are not suitable for very fine-grained typing.

New York Times consists of 500 random sentences from the New York Times corpus (Sandhaus, 2008), year 2007; we selected only sentences that contained at least one named entity according to the Stanford CoreNLP 4.4.1 tool.

CoNLL. We sampled 500 sentences from CoNLL (Tjong Kim Sang and De Meulder, 2003), a collection of newswires with manually annotated entities with CG types labels. We directly used the annotations in our evaluation. The sentences tend to be short and sometimes non-verbal (e.g., "Jim Grabb (U.S.) vs. Sandon Stolle (Australia)"). Most entities are prominent and likely to be found in our KB (and the one of existing methods).

Twitter. We collected the first 100 tweets with named entities retrieved by the Twitter API.

Type system. FINET's type system consists of more than 16k types with top categories persons, locations and organizations. We used the mapping between these top categories and WordNet described in Sec. 3.2. Hyena and Pearl use 505 and 200 WordNet types, resp., which is significantly smaller. To compare the performance across different granularities, we classified each type as coarse-grained (CG), fine-grained (FG) or super fine-grained (SFG). The CG types were ⟨*artifact*-1⟩, ⟨*event*-1⟩, ⟨*person*-1⟩, ⟨*location*-1⟩ and ⟨*organization*-1⟩. The FG types were those included in Pearl. All remaining types were considered SFG.

Labeling. All extractions by all systems were independently evaluated by two labelers. We adopted a pessimistic view, i.e., we treat an extraction as correct only if it was labeled correct by both labelers. The Cohen's kappa measure ranged 0.54–0.86, indicating a substantial agreement.

4.2 Results

Description of Tab. 3. Our results are summarized in Tab. 3. When a method did not produce a type of the considered granularity but a more fine-grained type, we selected its closest hypernym. For each configuration, the table shows the number of named entities for which types have been extracted, the total number of extracted types (more than one distinct type per named entity for some methods), the total number of correct types, and the precision (P). The number of named entities for which types have been found and the total number of correct extractions can be seen as a loose measure of recall. It is difficult to estimate recall directly for FG and SFG types since some entities may be associated with either no or multiple such types. To gain more insight, we show the number of correct distinct types, and the average depth (shortest path from ⟨*entity*-1⟩ in WordNet) for both correct FG and correct SFG types. Finally, we list the Cohen's kappa inter-annotator agreement measure for each method.

System	Coarse-Grained (CG)			Fine-Grained (FG)			Super Fine-Grained (SFG)			Distinct	Avg. Depth		Cohen's
	Enti-ties	Total types	Correct types *(P)*	Enti-ties	Total types	Correct types *(P)*	Enti-ties	Total types	Correct types *(P)*	types	FG	SFG	kappa
New York Times (500 sentences)													
FINET	992	992	872 *(87.90)*	616	631	457 *(72.42)*	319	329	233 *(70.82)*	191	5.96	7.25	0.60
FINET (w/o KB l.)	992	992	872 *(87.90)*	598	613	436 *(71.13)*	294	304	204 *(67.11)*	174	5.98	7.18	0.58
Hyena	895	1076	779 *(72.40)*	770	1847	522 *(28.26)*	518	775	160 *(20.65)*	127	5.79	6.98	0.74
Pearl (hard)	15	15	5 *(33.33)*	2	2	0	–	–	– *(–)*	1	–	–	0.54
CoNLL (500 sentences)													
FINET	1355	1355	1355 *(1.0)*	1074	1086	876 *(80.66)*	668	679	510 *(75.11)*	136	6.09	7.38	0.62
FINET (w/o KB l.)	1355	1355	1355 *(1.0)*	1075	1087	869 *(79.94)*	661	672	498 *(66.13)*	134	6.06	7.35	0.62
Hyena	1162	1172	1172 *(1.0)*	1064	2218	1329 *(59.92)*	719	944	268 *(28.39)*	103	5.89	6.57	0.69
Pearl (hard)	18	18	18 *(1.0)*	8	11	5 *(45.45)*	–	–	– *(–)*	7	5.6	–	0.74
Twitter (100 tweets)													
FINET	135	135	123 *(91.11)*	103	104	69 *(66.35)*	54	54	33 *(61.11)*	40	6.25	7.64	0.58
FINET (w/o KB l.)	135	135	123 *(91.11)*	104	105	65 *(61.90)*	56	56	30 *(53.57)*	40	6.14	7.6	0.55
Hyena	125	146	105 *(71.91)*	117	280	75 *(26.79)*	91	129	21 *(16.28)*	42	6.11	6.19	0.67
Pearl (hard)	10	10	5 *(50.00)*	3	4	1 *(25.00)*	–	–	– *(–)*	3	6	–	0.86

Table 3: Summary of results

Discussion. First note that Pearl extracted significantly fewer types than any other system. Pearl does not support SFG. For CG and FG, we conjecture that its database, which was generated from Wikipedia, did not reflect the syntactic structure of the sentences in our datasets. This finding strengthens the case for the use of heterogeneous sources in semi-supervised methods.

Hyena performed better than Pearl and in many cases extracted the largest number of types. Hyena tended to extract multiple types per named entity and mostly at least one FG type. This more recall-oriented approach, and its context-unaware use of supervision, significantly reduced its precision.

FINET had significantly higher precision across all settings, especially for SFG types (almost three times more than Hyena). One reason for this is that FINET is conservative: We provide more than one type per named entity only if the types were explicit. In all other cases, our type selection phase produced only a single type. FINET extracted the largest number of correct SFG types on each dataset. Hyena extracted more FG types, but with a significantly lower precision. The average depth of correct FG and SFG types in FINET was higher than that of Pearl and Hyena. FINET also tended to use more distinct correct types (191 in NYT vs. 127 for Hyena). Again, this more fine-grained typing stemmed from FINET's use of multiple extractors, many of which do not rely on supervision.

Note that FINET also has higher precision for CG. As stated, FINET makes use of the Stanford NER to extract CG types (except in CoNLL, were we used the manual labels), and respects these types for its FG and SFG extractions. Hyena has lower precision for CG types because it sometimes outputs multiple CG types for a single named entity. To ensure a fair comparison, for CoNLL we indirectly used the gold labels for Pearl and Hyena by discarding all types with an incorrect CG type.

FINET's extractors. Tab. 4 shows individual influence of each of FINET's extractors in the NYT dataset with KB lookups. The table shows the number of entities typed by each extractor and the precision of the resulting types after type selection. The mention-based extractor was the most precise and also fired most often, mainly due to locations. The pattern-based extractor also had a good precision and tended to fire often. The first three extractors, the more explicit ones, generated more than half of the extracted types; this indicates that explicit type extractors are important. There were also a substantial fraction of implicit types, covered by the corpus-based extractor. The verb-based extractor had the lowest precision, mostly because of the noisiness and incompleteness of its underlying resources (such as the (verb,type)-repository). We expect overall precision to increase if this extractor is removed. However, this would hinder FINET to infer types from verbs. Instead, we believe a better direction is to improve the underlying resources.

Error analysis. One source of error for FINET were incorrect CG labels. When CG labels were correct (by Stanford NER), the precision of FINET for FG types increased to more than 70% for all datasets. When FG labels were correct, the pre-

Last used extractor	Entities	P
Pattern-based	180	71.11
Mention-based	219	82.65
Verb-based	47	48.94
Corpus-based	205	64.39

Table 4: Per-extractor performance NYT

cision of SFG labels exceeded 90%.

Incompleteness of and noise in our underlying resources also affected precision. For example, some types in WordNet have missing hypernyms, which reduced recall; e.g., sponsor in WordNet is a person but cannot be an organization. WordNet is also biased towards US types (e.g., *supreme court* only refers to the US institution). Our repositories of verbs and their argument types are incomplete and noisy as well. Finally, errors in the KB affected both KB lookups and our corpus-based extractor. One example of such errors are temporal discrepancies; e.g., a person who used to be a $\langle$*player*-1$\rangle$ may now be a $\langle$*coach*-1$\rangle$. The KB types are also noisy, e.g., many soccer players in Yago2 are typed as $\langle$*football_player*-1$\rangle$ and the United Nations is typed as a $\langle$*nation*-1$\rangle$.

Finally, the type selection phase of FINET introduced mistakes (i.e., even when the correct type was a candidate, type selection failed to select it). This is especially visible in the verb-based extractor, which may produce a large number of candidates and thus makes type selection difficult.

Hyena mainly suffered from the general problems of supervised systems. For instance, since $\langle$*graduate*-1$\rangle$ or $\langle$*region*-1$\rangle$ are highly frequent in the KB, many persons (locations) were incorrectly typed as $\langle$*graduate*-1$\rangle$ ($\langle$*region*-1$\rangle$). Errors in the KB also propagate in supervised system, which may lead to "contradictory" types (i.e., an entity being typed as both $\langle$*person*-1$\rangle$ and $\langle$*location*-1$\rangle$).

5 Related Work

The NET problem is related to taxonomy induction (Snow et al., 2006; Wu et al., 2012; Shi et al., 2010; Velardi et al., 2013) and KB construction (Lee et al., 2013; Paulheim and Bizer, 2014; Mitchell et al., 2015), although the goals are different. Taxonomy induction aims to produce or extend a taxonomy of types, whereas KB construction methods aim to find new types for the entities present in a KB. In both cases, this is done by reasoning over a large corpus. In contrast, we are interested in typing each named entity mention individually using an existing type system. FINET draws from ideas used in taxonomy induction or KB construction. Existing systems are either based on patterns or the distributional hypothesis; these two approaches are discussed and compared in (Shi et al., 2010). In FINET, we make use of patterns (such as the ones of Hearst (1992)) in most of our extractors and of the distributional hypothesis in our corpus-based extractor.

Yahya et al. (2014) developed a semi-supervised method to extract facts such as "president"("Barack Obama", "US"), in which the relation acts as a type. FINET differs in that it supports implicit types and produces disambiguated types.

A number of NET systems have been proposed which make use of a predefined type hierarchy. Lin et al. (2012) proposes a semi-supervised system that uses relational patterns to propagate type information from a KB to entity mentions. Similarly, the subsequent Pearl system (Nakashole et al., 2013) is based on a corpus of typed relation patterns. An alternative approach is taken by supervised methods, which train classifiers based on linguistic features (Fleischman and Hovy, 2002; Rahman and Ng, 2010; Ling and Weld, 2012). Both Yosef et al. (2013) and Ling and Weld (2012) use Wikipedia and a KB to generate automatic training data. FINET is less reliant on a KB or training data than the above methods, which improves both precision (no bias against KB types) and recall (more fine-grained types supported).

Our type selection phase is based on WSD (Navigli, 2012), a classification task where words or phrases are disambiguated against senses from some external resource such as WordNet. Supervised WSD systems (Dang and Palmer, 2005; Dligach and Palmer, 2008; Chen and Palmer, 2009; Zhong and Ng, 2010) use a classifier to assign such senses, mostly relying on manually annotated data. KB methods (Agirre and Soroa, 2009; Ponzetto and Navigli, 2010; Miller et al., 2012; Agirre et al., 2014; Del Corro et al., 2014) use of a background KB instead.

6 Conclusion

We presented FINET, a system for fine-grained typing of named entities in context. FINET generates candidates using multiple extractors, ranging from explicitly mentioned to implicit types, and subsequently selects the most appropriate. Our experimental study indicates that FINET has significantly better performance than previous methods.

References

Eneko Agirre and Aitor Soroa. 2009. Personalizing pagerank for word sense disambiguation. In *Proceedings of EACL*, pages 33–41.

Eneko Agirre, Oier Lopez de Lacalle, and Aitor Soroa. 2014. Random walks for knowledge-based word sense disambiguation. *Computational Linguistics*, 40(1):57–84.

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The berkeley framenet project. In *Proceedings of ACL*, pages 86–90.

Satanjeev Banerjee and Ted Pedersen. 2003. Extended gloss overlaps as a measure of semantic relatedness. In *Proceedings of IJCAI*, pages 805–810.

Jinying Chen and Martha Palmer. 2009. Improving english verb sense disambiguation performance with linguistically motivated features and clear sense distinction boundaries. *Language Resources and Evaluation*, 43(2):181–208.

Hoa Trang Dang and Martha Palmer. 2005. The role of semantic roles in disambiguating verb senses. In *Proceedings of ACL*, pages 42–49.

Luciano Del Corro and Rainer Gemulla. 2013. Clausie: clause-based open information extraction. In *Proceedings of WWW*, pages 355–366.

Luciano Del Corro, Rainer Gemulla, and Gerhard Weikum. 2014. Werdy: Recognition and disambiguation of verbs and verb phrases with syntactic and semantic pruning. In *Proceedings of EMNLP*, pages 374–385.

Dmitriy Dligach and Martha Palmer. 2008. Improving verb sense disambiguation with automatically retrieved semantic knowledge. In *Proceedings of ICSC*, pages 182–189.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of ACL*, pages 363–370.

Tiziano Flati and Roberto Navigli. 2013. Spred: Large-scale harvesting of semantic predicates. In *Proceedings of ACL*, pages 1222–1232.

Michael Fleischman and Eduard Hovy. 2002. Fine grained classification of named entities. In *Proceedings of COLING*, pages 1–7.

Patrick Hanks. 1996. Contextual dependency and lexical sets. *International Journal of Corpus Linguistics*, 1(1):75–98.

Marti A. Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of COLING*, pages 539–545.

Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61.

Johannes Hoffart, Dragan Milchevski, and Gerhard Weikum. 2014. Stics: Searching with strings, things, and cats. In *Proceedings of SIGIR (demo)*, SIGIR '14, pages 1247–1248.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: The 90 In *Proceedings of HLT-NAACL (Companion Volume)*, pages 57–60.

Karin Kipper, Anna Korhonen, Neville Ryant, and Martha Palmer. 2008. A large-scale classification of English verbs. *Language Resources and Evaluation*, 42(1):21–40.

Shari Landes, Claudia Leacock, and Randee I. Tengi, 1998. *Building Semantic Concordances*. MIT Press.

Taesung Lee, Zhongyuan Wang, Haixun Wang, and Seung-won Hwang. 2013. Attribute extraction and scoring: A probabilistic approach. In *Proceedings of ICDE*, pages 194–205.

Beth Levin. 1993. *English Verb Classes and Alternations: A Preliminary Investigation*. University of Chicago Press.

Thomas Lin, Mausam, and Oren Etzioni. 2012. No noun phrase left behind: Detecting and typing unlinkable entities. In *Proceedings of EMNLP*, pages 893–903.

Xiao Ling and Daniel S. Weld. 2012. Fine-grained entity recognition. In *In Proceedings of AAAI*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Tristan Miller, Chris Biemann, Torsten Zesch, and Iryna Gurevych. 2012. Using distributional similarity for lexical expansion in knowledge-based word sense disambiguation. In *Proceedings of COLING*, pages 1781–1796.

Tom Mitchell, William Cohen, Estevam Hruscha, Partha Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohammad, Ndapa Nakashole, Emmanouil Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. 2015. Never-ending learning. In *Proceedings of AAAI*, pages 2302–2310.

Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. 2012. Patty: A taxonomy of relational patterns with semantic types. In *Proceedings of EMNLP*, pages 1135–1145.

Ndapandula Nakashole, Tomasz Tylenda, and Gerhard Weikum. 2013. Fine-grained semantic typing of emerging entities. In *Proceedings of ACL*, pages 1488–1497.

Roberto Navigli. 2012. A quick tour of word sense disambiguation, induction and related approaches. In *Proceedings of SOFSEM*, pages 115–129.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Heiko Paulheim and Christian Bizer. 2014. Improving the quality of linked data using statistical distributions. *IJSWIS*, 10(2):63–86.

Simone Paolo Ponzetto and Roberto Navigli. 2010. Knowledge-rich word sense disambiguation rivaling supervised systems. In *Proceedings of ACL*, pages 1522–1531.

Randolph Quirk, Sidney Greenbaum, Geoffrey Leech, and Jan Svartvik. 1985. *A Comprehensive Grammar of the English Language*. Longman.

Altaf Rahman and Vincent Ng. 2010. Inducing fine-grained semantic classes via hierarchical and collective classification. In *Proceedings of COLING*, pages 931–939.

Marta Recasens, Marie C. de Marneffe, and Christopher Potts. 2013. The Life and Death of Discourse Entities: Identifying Singleton Mentions. In *Proceedings of HLT-NAACL*, pages 627–633.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1988. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699.

Magnus Sahlgren. 2008. The distributional hypothesis. *Italian Journal of Linguistics*, 20(1):33–54.

Evan Sandhaus. 2008. The New York Times Annotated Corpus. *Linguistic Data Consortium, Philadelphia*, 6(12).

Shuming Shi, Huibin Zhang, Xiaojie Yuan, and Ji-Rong Wen. 2010. Corpus-based semantic class mining: Distributional vs. pattern-based approaches. In *Proceedings of COLING*, pages 993–1001.

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. 2006. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of COLING-ACL*, pages 801–808.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing with compositional vector grammars. In *Proceedings of ACL*, pages 455–465.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of HLT-NAACL*, pages 142–147.

Paola Velardi, Stefano Faralli, and Roberto Navigli. 2013. Ontolearn reloaded: A graph-based algorithm for taxonomy induction. *Computational Linguistics*, 39(3):665–707.

Ralph Weischedel and Ada Brunstein. 2005. BBN Pronoun Coreference and Entity Type Corpus. Technical report, Linguistic Data Consortium.

Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q. Zhu. 2012. Probase: A probabilistic taxonomy for text understanding. In *Proceedings of SIGMOD*, pages 481–492.

Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, and Gerhard Weikum. 2013. Robust question answering over the web of linked data. In *Proceedings of CIKM*, pages 1107–1116.

Mohamed Yahya, Steven Euijong Whang, Rahul Gupta, and Alon Halevy. 2014. Renoun: Fact extraction for nominal attributes. In *Proceedings of EMNLP*, pages 325–335.

Mohamed Amir Yosef, Sandro Bauer, Johannes Hoffart, Marc Spaniol, and Gerhard Weikum. 2012. HYENA: Hierarchical Type Classification for Entity Names. In *Proceedings of COLING*, pages 1361–1370.

Mohamed Amir Yosef, Sandro Bauer, Johannes Hoffart, Marc Spaniol, and Gerhard Weikum. 2013. HYENA-live: Fine-Grained Online Entity Type Classification from Natural-language Text. In *Proceedings of ACL*, pages 133–138.

Zhi Zhong and Hwee Tou Ng. 2010. It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of ACL: System Demonstrations*, pages 78–83.