A Appendix

A.1 Encoder Details

We use the GRU [Cho *et al.*, 2014] and the scaled dot-product attention [Vaswani *et al.*, 2017] to learn embeddings for sentences.

For the i-th sentence of F, s_i, we first convert each word $w_{i,j}$ into a word embedding vector $\mathbf{w}_{i,j} \in \mathbb{R}^{d_w}$ by the pre-trained word embedding matrix. Then we use the GRU to capture the dependencies among words: $\mathbf{x}_{i,j} = \text{GRU}(\mathbf{w}_{i,j})$. Here $\mathbf{x}_{i,j} \in \mathbb{R}^d$ is the hidden state of GRU. Since a word cannot well describe the instance of a CE, we then perform sentence-level aggregation. We utilize the scaled dot-product attention to capture informative words and aggregate these hidden states into a single embedding accordingly, which is faster and more space-efficient than the additive attention. A slight difference is that in order to better capture important sentence-level features, we compute the query vector by a max-pooling operation over these hidden states. Specifically, we use $\{\mathbf{x}_{i,j}\}$ to construct a matrix $\mathbf{X}_i \in \mathbb{R}^{d \times m_i}$ and then the sentence-level fact embedding $\mathbf{f}_i$ is computed as:

$$\mathbf{f}_i = \mathbf{X}_i(\text{softmax}(\mathbf{K}_i{}^\intercal \mathbf{q}_i/\sqrt{d})),$$

where $\mathbf{K}_i \in \mathbb{R}^{d \times m_i}$, a linear transformation of $\mathbf{X}_i$, is the keys of words, and $\mathbf{q}_i$ is the query vector which is computed by a max-pooling operation over $\mathbf{X}_i$:

$$\mathbf{q}_i[a] = \max_{1 \le j \le m_i} (\mathbf{x}_{i,j}[a]), \;\; a \in \{1, \ldots, d\},$$

where $\mathbf{q}_i[a]$ denotes the a-th dimension of $\mathbf{q}_i$. We use a max-pooling operation to capture the most important feature for each dimension and obtain a sentence-level query. Here $\mathbf{q}_i$ holds the sentence-level information and $\mathbf{K}_i{}^\intercal \mathbf{q}_i$ evaluates the correlation between each word and the sentence. In this way, the words that contribute most to the sentence can be captured.

A.2 Training

We optimize the RL module, and the encoder and predictor alternately. Specifically, the encoder and predictor are updated every batch using the cross-entropy loss function. For the legal agent, we employ the asynchronous advantage actor-critic (A3C) algorithm [Mnih *et al.*, 2016] to train it. A3C is an efficient and effective method for training reinforcement learning algorithms. In order to improve the exploration of policy π and discourage premature convergence to suboptimal policies, an entropy regularization of policy π is employed. The resulting gradients of the policy π and value function are respectively:

$$\begin{gathered} \nabla^t_{\theta_a} \log \pi^t_{\theta_a}(a^t)(R^t - v^t_{\theta_v}) + \beta \nabla_{\theta_a} H(\pi^t_{\theta_a}), \\ \nabla^t_{\theta_v}(R^t - v^t_{\theta_v})^2, \end{gathered} \tag{9}$$

where $R^t = \sum_{i=0}^{T-1} \gamma^i r^{t+i}$ and γ is a discount factor. We update the agent when t grows to the maximum step number T, where T is the preset number of total sentences that the agent selects for all CE types. Algorithm 1 shows the pseudocode of the training process. Moreover, we pre-train the encoder and predictor since the training of the RL module is slow and unstable.

Algorithm 1 Pseudocode for training CECP.

Parameter: Encoder and Predictor θ; Legal Agent θ_a, θ_v
Data: Cases set $\{(F, y)\}$; CEs set $\{e_{p,c}\}$

```
Initializing the model.
while Training do
    Encode a batch fact descriptions {F} and CEs set {e_{p,c}}.
    for i ← 0 to batch size do
        Take a fact description F.
        for t ← 1 to T do
            p = 1 + [(t − 1) mod P];
            Perform an action a^t to select one sentence for CE type p according to the policy π (Eqs. (2) ∼ (5));
            Run predictor to obtain pred^t, and compute a reward r^t according to Eq. (7).
        end for
        Update the agent according to Eq. (9).
    end for
    Run predictor according to the final selected sentences and the CEs, and obtain the final prediction, then update the encoder and predictor.
end while
```

A.3 Implementation Details

We provide details of our implementation and experimental setup in this section to help reproduce the findings in this work. We will release the codes and datasets.

Dataset Preprocessing

Since the CAIL dataset is a multi-label dataset, we perform preprocessing following [Xu *et al.*, 2020]. Concretely, we extract the fact descriptions, charges, as well as applicable law articles from CAIL. For fact descriptions, we find that those cases whose fact descriptions are fewer than 30 words do not provide any useful information with regard to charge, so we first filter out these cases. For charges, since we aim to explore the effectiveness of CEs in charge prediction, we filter out those cases with multiple charges, and remove charges with associated cases less than 100 (also remove the associated cases). It should be noted that although our model does not exploit the information of law articles[4], we extract them since they are used in some baselines (LADAN, FLA). We perform preprocessing on the law articles in the same way as we process the charges.

CEs Construction

There is no publicly available CEs knowledge in previous works for charges prediction. Considering we use China judgment datasets in our experiments, we collect CEs descriptions of the charges from zuiming.net[5]. The CEs system of China considers four CE types, subject element, subjective element, object element, and objective element ($P = 4$). In total, we have collected the CEs of 168 charges, covering all the charges involved in both Criminal and CAIL datasets. It

[4]Constitutive elements are judicial interpretations that are not included in law articles.

[5]https://www.zuiming.net/

is found that the average lengths for the above four CE types are 61.5, 64.0, 107.4, and 269.4 words, respectively.

Baseline Settings

There are two styles of legal knowledge for the legal knowledge based baselines. One is the manually defined legal attributes, which are 0/1 values indicating whether a certain legal attribute (e.g., the criminal has the act of violence) exists. The other is law articles in the Criminal Law, where a law article consists of a numeric label and a piece of text.

For LADAN, we only apply it on the CAIL dataset because the Criminal dataset does not provide an exact numeric label of law article for a case, which is pivotal for the Graph Distillation Operator in LADAN. For BERT, we use the pre-trained model parameters released by [Zhong *et al.*, 2019], which are trained on Chinese criminal cases.

Word Embedding Pre-training

To pre-train word embeddings, we use the THULAC tool[6] for word segmentation since the raw data is written in Chinese. Then we train a Skip-Gram model on the segmented data to obtain word embeddings. The word embedding size is set as 200 and the frequency threshold is set to 25.

Hyperparameters Settings

For our model and those baselines with a hierarchical-structure encoder (HARNN, FLA, LADAN), we set fact descriptions' maximum sentence length to 32 words and maximum document length to 64 sentences. We set the maximum document length as 512 words for the other models. Within this setting, the average number of words contained in the fact descriptions is almost the same for all models. For CEs, considering that the text lengths and the amounts of information of different CE types are different, we set the maximum length of subject element, subjective element, object element, and objective element as 100, 100, 200, and 400 words, respectively. We set all latent states to 128 and batch size to 64.

For the other hyperparameters in CECP, we select them according to the performance on the validation set. Specifically, γ, β and λ are set to 0.95, 0.015 and 0.1, respectively. The n_p, which is the total number of selected sentences for CE type p, is set as 6 ($p = 1$), 3 ($p = 2$), 3 ($p = 3$), 6 ($p = 4$), respectively. This means that the maximum step T is set to 18. For the other hyperparameters in baselines, we follow the settings described in their papers.

Training Details

For baselines, we use the Adam optimizer with a learning rate of 10^{-3} to train them for 16 epochs, which is a popular setting in mainstream works [Xu *et al.*, 2020].

Since the training of the RL module is slow and unstable, we first pre-train the encoder and predictor of CECP. In this stage, we neglect the sentence selection process. Specifically, we set $n^t = n$ for Eqs. (2) $\sim$ (5), i.e., all sentences in the fact description are treated as the selected sentences. We use the Adam optimizer with a learning rate of 10^{-3} to train the encoder and predictor for 12 epochs. After that, we alternately optimize the modules in CECP in a batch-manner (Pseudocode in Algorithm 1) for 4 epochs, where the optimizer for the encoder and predictor is Adam with a learning rate of 10^{-3}, and for the RL module, it is the ShareRMSProp [Mnih *et al.*, 2016] with a learning rate of 10^{-4}. Note that we train the encoder and predictor with the Adam optimizer for a total of 16 epochs, which is consistent with that of baselines.

For all models, we train and test each of them 5 times, and finally report the average results.

Experimental Environment

Our model and baselines are implemented by PyTorch and run on Ubuntu Linux 18.04. (Hardware used: 16-core Intel Xeon CPU E5-2620 v4 at 2.10 GHz, NVIDIA GeForce GTX 1080 Ti).

[6]http://thulac.thunlp.org/

A.4 Numerical Results of Ablation Study

This table shows the numerical results corresponding to Fig 3(a).

	Acc.	MP	MR	F1
No-PFI	0.8596	0.8527	0.8589	0.8492
Order-1	0.8572	0.8497	0.8547	0.8452
Order-2	0.8618	0.8531	0.8543	0.8472
CECP	**0.8646**	**0.8563**	**0.8599**	**0.8520**

Reference

[Zhong *et al.*, 2019] Haoxi Zhong, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. Open chinese language pre-trained model zoo. Technical report, 2019.