An NLP and LLM based Form Filling Method

by

Songzhe Li

Supervised by

Dr. Lingqiao Liu

In fulfilment of the requirements for the degree of

Bachelor of Computer Science (Honours)

June 2023

Faculty of Science, Engineering and Technology

The University of Adelaide

Contents

List of Figures

Declaration

Songzhe Li 15/6/2023

Acknowledgements

I owe my profound gratitude to my supervisor, Lingqiao Liu. His wisdom, patience, and steadfast guidance have been instrumental in shaping my academic journey. The knowledge shared and the inspiration imparted will continue to light my path ahead.

My deepest appreciation goes to my parents. Their boundless love, relentless support, and enormous sacrifices have carved my path and fueled my achievements. The opportunities and growth I've experienced can be directly attributed to their unwavering belief in me.

I extend heartfelt thanks to my friends who have helped me along the trip. Each insightful critique, meaningful discussion, and offered assistance significantly enriched my research journey. Your companionship made this challenging journey more enjoyable and rewarding.

To everyone who extended their support in my academic venture, your encouragement, and faith have left an indelible mark on my journey. Your backing has been my guiding light, leading me through the challenging maze of academia.

"Per Aspera Ad Astra"

This journey, with its hurdles and triumphs, echoes these ancient words. A sincere thank you to all who have guided me through the rough seas and cheered me towards my personal stars. Your unwavering support and belief have truly brought this timeless phrase to life.

Abstract

This study presents a novel approach to the automation of PDF form filling utilizing synergistic capabilities of Natural Language Processing (NLP) and Large Language Models (LLMs). The research question addressed is ”How can NLP and LLMs be effectively employed to automate PDF form filling using pre-existing user information?” Traditional NLP techniques (GloVe) and cutting-edge LLMs (GPT3.5) are integrated in our system to automate form filling tasks. Our investigation provides a comprehensive review of vector semantics methods in NLP including Word2Vec, LSA, and GloVe, their principles, advantages, and drawbacks, and existing form filling methodologies with an emphasis on their strengths and weaknesses. Subsequently, the LLMs were scrutinized, focusing on their underlying principles, merits, and demerits. Following the examination of relevant literature and technologies, a software implementation is proposed. The system functions through a user-provided input-output form setup and leverages a combination of word embeddings, GPT model, and a unique dictionary method for optimal output form filling. In a practical application scenario of a rental application, our system achieved an overall accuracy rate of 74.72% with a total response time of 108.78 seconds. The paper also highlights the limitations while outlining potential directions for future development considering the enhancement of performance and ethical issues. The findings underscore the viability of NLP and LLMs integration for improving efficiency in data entry tasks.

Chapter 1

Introduction

1.1 Research Problem

The digitization of form filling in PDF formats is a vital and complex task with expansive applications across various sectors such as healthcare, administration, and job or rental application processes. This routine and repetitive process can be resource-intensive, demanding considerable time, labor, and attention to detail to prevent errors. This becomes particularly cumbersome when users need to fill similar new forms repeatedly, which often contain redundancies that make manual filling tedious and inefficient.

Imagine a scenario where a user needs to fill multiple rental applications within a short span of time. While some form fields may be unique, a majority of them - such as personal identification information, income details, and previous rental history - would likely be repetitive. Yet, despite this repetitiveness, the user is required to input the same details manually into each form, which not only consumes a significant amount of time but also introduces the risk of errors.

This context highlights a key challenge that many individuals and organizations face when dealing with PDF forms: the absence of an efficient and accurate automation process that can fill forms using pre-existing user information. Traditional auto-fill tools are often ill-equipped to handle the nuances of language, context, and intent inherent in form fields.

The primary research question, therefore, is: "How can Natural Language Processing (NLP) and Large Language Models (LLMs) be synergistically employed for automating the filling of PDF forms using pre-existing user information?" This research aims to investigate how these advanced language technologies can be harnessed to interpret the semantic intricacies of form fields, generate relevant, context-specific responses, and thus streamline the process of form filling. The goal is to design and implement a system that not only understands the form prompts and their inherent semantics but also accurately fills them with the appropriate content based on previously provided user information.

1.2 Research Purpose and Goal

The central purpose of this research is to harness the power of Natural Language Processing and Large Language Models (LLMs) to address a significant challenge faced by users in the digital landscape - the lack of effective tools for automatically filling information in PDF forms. Traditional auto-fill tools, usually designed for web forms, often fall short when applied to PDF forms, creating a gap in the automation of form filling processes. This project aims to fill this gap by developing a novel system capable of auto-filling PDF forms, utilizing state-of-the-art LLMs.

The ambition extends beyond merely developing an automated system, seeking to substantially decrease the time and effort users expend in entering repetitive information across multiple PDF forms. By making use of prior user information, the system aims to intelligently and accurately fill similar new forms, thus enhancing efficiency and minimizing user burden. Such automation could substantially alleviate labor-intensive form filling tasks, thereby freeing up human resources for other essential tasks.

The proposed system will leverage the capabilities of LLMs, which have demonstrated remarkable prowess in understanding, interpreting, and generating human language. This technology will be applied to comprehend the context and intent of form fields, and then autonomously fill these fields with accurate, context-appropriate responses. This two-pronged approach is expected to overcome the challenges inherent in traditional form filling tasks, such as language nuances, contextual understanding, and form design variability.

The ultimate goal of this research is threefold. Firstly, to develop an innovative system that can automate the process of filling in PDF forms, overcoming the limitations of existing auto-fill tools. Secondly, to enhance user experience and productivity by significantly reducing the time and effort involved in filling repetitive information across multiple PDF forms. Lastly, to explore and pioneer a new frontier in the field of Natural Language Processing by synergistically combining traditional NLP techniques with the most recent advancements in Large Language Models. This integration aims to open new possibilities in form filling automation, demonstrating the combined capabilities of NLP and LLMs in understanding context and intent and generating accurate, context-appropriate responses. This research ultimately seeks to offer valuable insights into the way users interact with PDF forms.

1.3 Report Overview

This report commences with the research question and defines its purpose and goals. The following chapter provides an in-depth literature review of Vector Semantics Methods such as Word2Vec, LSA, and GloVe, as well as Machine Learning Based Form Filling Methods. The use of Large Language Models like GPT, LLaMa, and Alpaca is also examined, along with a rationale for choosing GPT for this project.

The subsequent section of the report deals with the software implementation aspect. It covers practical applications, presents the methodology, and gives details about the code implementation. This part also outlines key features of the implementation such as Cosine Similarity calculation speed, cost-effective use of GPT, and model selection.

The next chapter offers an experimental results section, discussing testing materials, evaluation metrics, and an extensive functionality and performance evaluation.

Thereafter, limitations of the current system are outlined, touching upon both software incapabilities and general restrictions. The report concludes with a look towards future development, focusing on performance enhancement and ethical issues. Please note that this overview gives a general direction of the report, and the main body will present a more detailed examination of these topics.

Chapter 2

Literature Review

2.1 Overview

This section provides a comprehensive examination of prominent methodologies within the field of Natural Language Processing (NLP), and their implications for the technology of automatic form filling. First, we delve into vector semantics methods, including Word2Vec, LSA, and GloVe. These methods have been widely employed for word representation, each boasting unique strengths and weaknesses, which we will expound on in detail. Following that, we explore existing form filling methods, including automatic form filling techniques in web browsers and those based on machine learning. While these techniques have to some extent addressed the automation of form filling, they remain with certain limitations. Finally, we investigate Large Language Models (LLMs) such as GPT, LLaMa, and Alpaca, conducting a thorough comparison of their principles, advantages, and drawbacks, so as to conclude why GPT is particularly suited for this project. The aim of this literature review is to equip readers with a well-rounded background knowledge, aiding in a deeper understanding of the challenges faced and approaches taken in this research.

2.2 Vector Semantics Methods

2.2.1 Word2Vec

Word2Vec, also known as word embeddings, is a powerful tool in the field of natural language processing (NLP) that generates word vectors from natural languages. It utilizes a single-layer neural network, specifically the Continuous Bag-of-Words (CBOW) model, to map a sparse word vector in one-hot form to a dense word vector in distributed form (Bernal et al., 2021). This transformation is essential as computers cannot directly comprehend human language, which is often abstract and symbolic.

Mathematical models, on the other hand, require numerical inputs, necessitating the conversion of words into numerical form or 'embedding' them in a mathematical space.

One of the significant advantages of Word2Vec is its ability to address the issue of data sparsity, a common problem in text analysis where text is uniquely numbered (Bernal et al., 2021). It takes into account the context of words, which enhances its performance over previous embedding methods. Additionally, Word2Vec operates at a faster pace due to its lower dimensionality compared to other approaches. Furthermore, Word2Vec is capable of exploiting higher-order co-occurrences and global relations, which is beneficial in modeling human cognitive tasks such as human word-learning and semantic memory (Altszyler et al., 2017). This ability to take advantage of indirect learning sets Word2Vec apart from other models like Latent Semantic Analysis (LSA).

However, Word2Vec is not without its limitations. It establishes a one-to-one relationship between words and vectors, which hampers its ability to effectively identify synonyms (Altszyler et al., 2017). Furthermore, despite its high versatility, Word2Vec, being a static approach, cannot be dynamically tuned for specific applications. Another notable drawback is that Word2Vec's performance decreases when the corpus is cleaned of out-of-domain documents. This is because Word2Vec does not shift its maximums in the out-of-domain documents discarding method, making it less sensitive to the reduction of topic variety (Altszyler et al., 2017).

In conclusion, while Word2Vec has proven to be a powerful tool for generating word vectors, it is not without its limitations. Future research should focus on addressing these limitations and exploring the potential of Word2Vec in other applications.

2.2.2 LSA

Latent Semantic Analysis (LSA) is a method in natural language processing that leverages vector representations for both words and documents, thereby establishing relationships between terms and documents based on the associations between different vectors. This process involves the application of statistical analysis on extensive text datasets, facilitating the extraction of words' contextual usage and meanings (Altszyler et al., 2017).

A notable strength of LSA lies in its capacity to mitigate the impact of synonyms via Singular Value Decomposition (SVD), which subsequently enhances the precision of post-processing tasks (Altszyler et al., 2017)). In contrast to the conventional vector space model, LSA projects words and documents into a latent semantic space, thereby eliminating certain "noise" present in the original vector space and improving the precision of information retrieval. Additional benefits of LSA encompass its ability to enhance feature robustness and remove noise through dimensionality reduction, its efficient utilization of redundant data, its unsupervised learning approach, and its adaptability to various languages (Heo and Choi, 2023).

Despite these advantages, LSA has disadvantages. It suffers from a lack of robust mathematical and statistical foundations, and the computational complexity associated with SVD is considerably high (Heo and Choi, 2023). Furthermore, the introduction of a new document necessitates the retraining of the model. A significant drawback of LSA is its disregard for the order of words within an article or sentence, which could result in the loss of crucial contextual information (Heo and Choi, 2023).

In summary, while LSA serves as a potent tool for discerning semantic relationships between words and documents, it is accompanied by certain limitations. Future research endeavors should aim to address these limitations and explore the potential applications of LSA in other domains.

2.2.3 GloVe

GloVe, an acronym for Global Vectors for Word Representation, is a tool that captures word representations based on global word frequency statistics. Each word is represented as a vector, and these vectors encapsulate the semantic differences between words (Pennington et al., 2014). GloVe combines both global and local statistical information of words to generate word vectors. It constructs a co-occurrence matrix, where each element represents the number of occurrences of a pair of words within a specific contextual window (Valentini et al., 2021).

GloVe's strength lies in its ability to capture both global and local word relationships, making it more effective at capturing semantic and syntactic word relationships (Valentini et al., 2021). This is a significant advantage over Word2Vec, which is trained based on local prediction (Bernal et al., 2021). For instance, in tasks such as gender-occupation and gender-name associations, GloVe has been found to perform comparably to other methods like Skip-gram and PMI-based metrics (Valentini et al., 2021). This ability to capture both local and global relationships allows GloVe to provide a more comprehensive representation of word semantics, which can be particularly useful in tasks that require a deep understanding of word relationships, such as text summarization or translation.

However, similar to Word2Vec, GloVe's word vectors are static, which means they cannot effectively handle synonyms and polysemous words. This static nature also means that the vectors cannot be dynamically adjusted for specific tasks or contexts (Pennington et al., 2014). This limitation can be particularly problematic in tasks that require a nuanced understanding of word meanings, such as sentiment analysis or text classification. For instance, the word "bank" can have different meanings depending on the context, and a static word vector may not be able to capture these nuances.

Despite these cons, GloVe's performance in tasks such as gender-occupation and gender-name associations have been found to be comparable to other methods like Skip-gram and PMI-based metrics (Valentini et al., 2021). Furthermore, in the context of matrix factorization, GloVe has been found to outperform vanilla matrix factorization techniques in both accuracy and fairness metrics (Wang, 2021).

In the context of the research question, "How can Natural Language Processing (NLP) and Large Language Models (LLMs) be synergistically employed for automating the filling of PDF forms using pre-existing user information?", GloVe's ability to capture both global and local word relationships makes it a suitable method for understanding and representing the semantics of the user information. Its performance in tasks involving associations between different types of words also suggests its potential effectiveness in understanding and representing the relationships between different pieces of user information.

2.3 Existing Form Filling Methods

2.3.1 Form Filling Methods on Web Browsers

The automatic form fill, or autofill, feature in web browsers is a significant productivity tool that predicts the field labels of a web form and automatically fills values in a new form based on the values previously filled for the same field in other forms (Bose, 2019). This feature is present in all major web browsers, including Google Chrome, and it enhances productivity as the user does not have to fill the same form field repeatedly for multiple forms (Google. (n.d.), 2023).

The technical principle behind the autofill feature involves predicting the field labels and automatically suggesting or filling the previously stored information for those fields, based on the user's historical data stored locally in the browser (Bose, 2019). A naïve approach for predicting the form labels is to keep a file with the extracted features and the predicted form labels for each field in each form. However, such an approach would not be scalable, given the billions of forms on the web. Major browsers like Chrome and Firefox use a combination of heuristic rules for this purpose. For instance, if the Id or name or label of the field HTML follows a specific regular expression, then the predicted field should be set to a specific value. However, such an approach may not give desired results for some kinds of forms where the Id may be ambiguous or for forms that are dynamic, such as forms behind a login or paywall (Bose, 2019).

The application of the autofill feature is widespread, especially in scenarios where users have to fill similar information in fields in multiple forms. For instance, when a user is shopping online and needs to input their shipping address on various platforms, the autofill feature can save time and increase efficiency by automatically filling in the required information.

The advantages of the autofill feature are evident. It increases the convenience and efficiency of users, especially those who have to fill similar information in fields in multiple forms. It also enhances productivity as the user does not have to fill the same form field repeatedly for multiple forms (Bose, 2019). There are certain drawbacks worth mentioning too. For example, suppose the autofill function fails to accurately forecast the label of a particular field. In that case, it may populate errone-ous data, leading to possible mistakes or misunderstandings. Additionally, there are privacy concerns, as the user's personal information is stored locally in the browser and used to predict and fill in information in various forms (Bose, 2019).

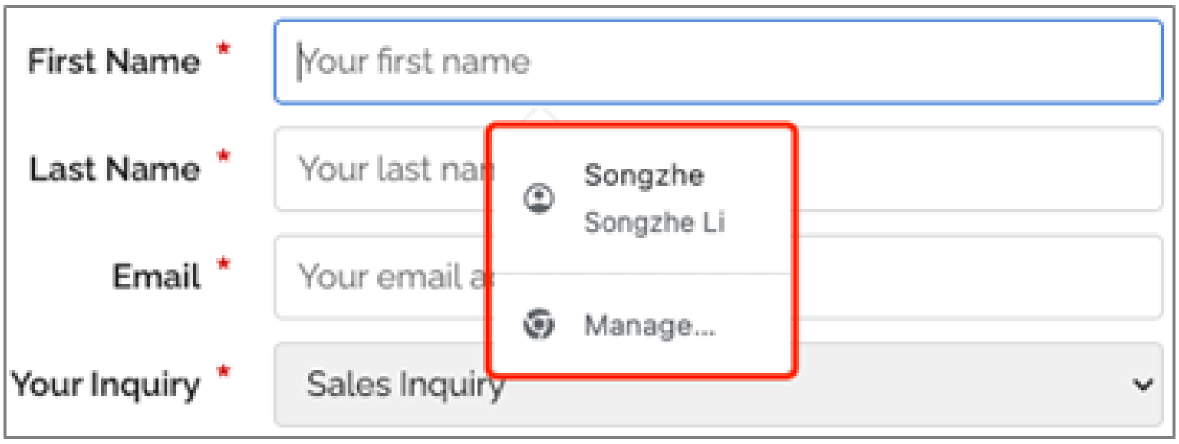


Figure 2.1: Autofill in Google Chrome Browser

2.3.2 Three Machine Learning based Form Filling Methods

In regard to machine learning-based methods for automatic form filling, we have conducted a detailed investigation into three approaches. They all employ deep learning techniques while having their own distinctions. This paragraph will provide a comprehensive overview of the technical details and advantages and disadvantages of these three methods.

Method 1

The "Design and Implement of Online Intelligent Form Filling System" by (Qiu and Dai, 2012) presents an innovative approach to form filling that leverages Excel templates. The system is designed to overcome the barriers of time and space during form filling, reduce the workload of form filling and aggregation, and improve work efficiency. The system is divided into five modules: template management, form management, friend management, user management, and system management.

The template management module allows users to manage Excel templates, including uploading, downloading, deleting, and publishing. The form management module manages web forms that are automatically parsed and generated from the user's uploaded Excel templates. It allows users to view filled forms and summarize all of them. The friend management module manages users' friends, allowing users to authorize all users or only a subset of their friends to fill out web forms. The user management module manages users in the system and provides a password modification function for all users. The system management module manages system information, such as system settings, system codes, and fundamental data, and also provides a data backup function.

It's worth mentioning that the system uses synonym table defined in advance to process fields with the same meaning but different names. This process can be quite tedious as it requires the user to define a synonym table in advance. The system's effectiveness is heavily dependent on the breadth of the synonym table, making it difficult to adapt to diverse forms. This approach contrasts with heuristic rule-based methods used in browser autofill systems, which typically use pattern recognition and field type identification to fill out forms (Qiu and Dai, 2012).

Method 2

In the paper "Field Label Prediction for Autofill in Web Browsers" by (Bose, 2019), the author presents a machine learning-based approach to autofill in web browsers. The system uses a combination of field type identification and pattern recognition to predict the most likely field labels. The system is trained on a large dataset of web forms, allowing it to learn common field labels and their associated patterns.

Specifically, the system's architecture includes a web service for predicting field labels based on features extracted from the HTML of web forms. The machine learning model is trained on the server and predicts the label of the form field in real time. This approach offers the advantage of easy model updates when new data is acquired. However, it's important to note that the actual form data remains in the web browser client for privacy reasons, and only the features related to the form and field details are sent to the server for label prediction.

The author also discusses the training process of the machine learning model. The model is trained using a dataset generated through a crowdsourced method with human labelers, which includes around 4000 values from commonly used web forms. The input features used for training include label, name, id, and URL. Each form field is trained with a separate binary classifier, although a multi-class classification approach can also be experimented with (Bose, 2019).

Nevertheless, the performance of the system heavily relies on the quality and variety of its training data. The author recognizes that training the model on a more comprehensive and diverse dataset can enhance its performance. Additionally, by integrating an ensemble of different approaches, a hybrid approach experiment can further boost the system's effectiveness. This approach offers a more flexible and adaptable solution compared to the synonym table-based method, as it can learn and adapt to new forms and field labels over time (Bose, 2019).

Method 3

In the study "Personal Information Classification Based on Deep Learning in Automatic Form Filling System" by (Wu et al., 2020), a deep learning-based system for automatic form filling is proposed. The system is designed with a comprehensive architecture that includes data collection, preprocessing, deep learning, and storage modules. The data collection module, or crawler, gathers necessary data from the internet, which is then cleaned and standardized by the preprocessing module. This step is crucial to ensure the quality of the data used for training the deep learning module (Wu et al., 2020).

The system relies on a deep learning model powered by the Convolutional Neural Network (CNN) to accurately identify and classify key information. Its training dataset comprises ample personal data, allowing it to predict the most probable field values in any form. This strategy provides a high degree of accuracy and adaptability, as the system can learn from and adjust to new forms and information over time (Wu et al., 2020).

However, as the same result got in method 2, this machine learning based form filling method also heavily relies on the training data. It also requires substantial computational resources to train and operate the deep learning model. The classified data is stored and retrieved when a user needs to fill in a form, allowing the system to adapt to user preferences over time.

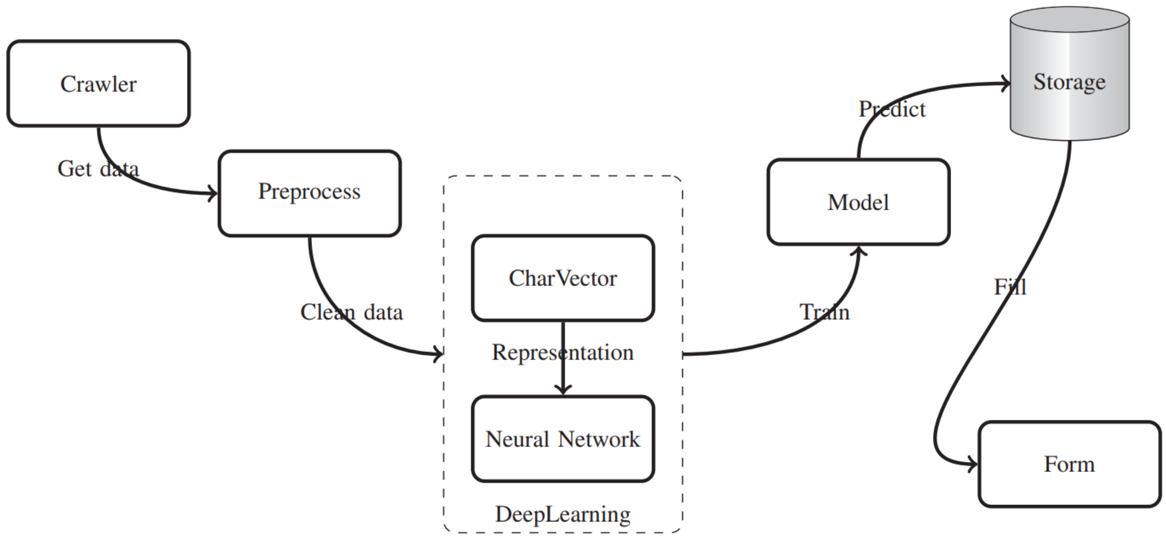


Figure 2.2: Personal Information Classification System architecture (Wu et al., 2020)

Conclusion

In conclusion, machine learning-based autofill technologies offer a more flexible and adaptable solution to form filling compared to heuristic rule-based methods. Despite holding great promise for improving the efficiency and accuracy of form filling in the future, machine learning-based autofill technologies also pose challenges. These include the requirement for large and varied training datasets, substantial computational resources, and potential issues with overfitting or underfitting.

2.4 Large Language Models

This section will introduce three popular large language models: GPT, LLaMa, and Alpaca. We will describe their characteristics and compare their advantages and disadvantages.

2.4.1 GPT

GPT-3.5 Turbo, a variant of the GPT-3 model, is a large language model (LLM) developed by OpenAI. It has been trained on a vast amount of text data using self-supervised learning, and it has demonstrated impressive capabilities in various applications (Jang and Kim, 2023). The model is composed of a significant number of parameters, which contribute to its ability to generate human-like text based on the input it receives.

One of the key features of GPT-3.5 Turbo is its ability to perform complex tasks by reasoning step-by-step (Poesia et al., 2023) introduced a class of tools for language models called guides, which use state and incremental constraints to guide the generation of the model. They demonstrated how a general system for logical reasoning, named LogicGuide, can be used as a guide for GPT-3.5 Turbo. The model can formalize its assumptions for LogicGuide and then ensure that its reasoning steps are sound. This feature significantly improves the performance of GPT-3.5 Turbo and reduces the interference of prior and current assumptions, a phenomenon known as content effects.

In addition to its reasoning capabilities, GPT-3.5 Turbo has shown impressive performance in domain-specific applications. For instance, (Jang and Kim, 2023) assessed the capabilities of GPT-3.5 Turbo in Traditional Korean Medicine (TKM) using the Korean National Licensing Examination for Korean Medicine Doctors. The model demonstrated impressive medical knowledge despite lacking medicine-specific training, indicating its potential in culturally-adapted medicine, specifically TKM, for clinical assistance, medical education, and medical research.

To sum up, GPT-3.5 Turbo is a powerful model that has demonstrated impressive capabilities in various applications, from logical reasoning to domain-specific tasks. Its ability to reason step-by-step and adapt to specific domains makes it a valuable tool in the field of artificial intelligence.

2.4.2 LLaMa

LLaMa, like the GPT series models, is a generative dialogue model that excels in tasks such as text classification and multiple-choice question answering (Ye et al., 2023). Its architecture, which comprises an embedding layer, multiple transformer blocks, and a language model head layer, is enhanced with several improvements such as Pre-normalization, SwiGLU activation, and Rotary Embeddings. These enhancements, coupled with a total parameter range from 7B to 65B, underscore LLaMa's complexity and potential for handling intricate tasks.

Despite these strengths, LLaMa faces challenges when applied to specific domains like law or medicine, primarily due to a deficiency in domain-specific knowledge and an inadequate capability to leverage such knowledge to address domain-related problems. To mitigate these issues, continual training of LLaMa on diverse corpora or with various supervised fine-tuning datasets is recommended (Ye et al., 2023).

Another limitation of LLaMa pertains to its language support. The model has been pre-trained on 1T to 1.4T tokens from publicly available corpora, predominantly in English and only a small fraction in other languages using Latin or Cyrillic scripts.

Consequently, LLaMa's ability to understand and generate text in languages that use non-Latin or non-Cyrillic scripts is limited (Ye et al., 2023).

On a positive note, LLaMa exhibits a significant advantage in terms of deployment. Its ability to be deployed on personal computers using llama.cpp (or similar) enables a more versatile and privacy-conscious utilization of LLMs across various domains, marking a significant stride in the field (Ye et al., 2023).

2.4.3 Alpaca

Alpaca is a large language model that has demonstrated its ability to solve a simple numerical reasoning problem in a human-interpretable way. It achieves near-perfect task performance by implementing a simple algorithm with interpretable variables (Wu et al., 2023). Furthermore, Alpaca uses this simple algorithm across a wide range of contexts and variations, which makes it a flexible and adaptable model for different applications. The alignment of neural representations with these variables is robust to changes in inputs and instructions, marking a first step toward deeply understanding the inner workings of large language models (Wu et al., 2023).

However, Alpaca also has its limitations. While it achieves good task performance, it does not always perform perfectly. For instance, the Alpaca (7B) model achieves 85% for randomly drawn triples of numbers on average (Wu et al., 2023). This suggests that there might be room for improvement in its performance.

In terms of its implementation, Alpaca has been designed for intermittent computing on energy-harvesting devices. It has a low overhead compared to existing systems, which makes it efficient for tasks that require computing resources (Maeng et al., 2019). Alpaca programs are composed of a sequence of user-defined tasks, and the Alpaca runtime preserves execution progress at the granularity of a task. This ensures that data remain consistent despite power failures, which is crucial for automating the filling of PDF forms using pre-existing user information (Maeng et al., 2019). However, Alpaca requires careful programming to ensure data consistency, which could pose a challenge for its deployment and use in practical applications (Maeng et al., 2019).

2.4.4 Conclusion

In conclusion, GPT, LLaMa, and Alpaca each have their unique strengths and weaknesses. GPT's strength lies in its pre-training on a large amount of data and its large number of parameters, but its overparameterized nature can be a limitation. LLaMa performs well across various tasks, but its performance can be limited in specific domains. Alpaca provides a low-overhead implementation and improves performance and memory footprint, but it has limitations in natively supporting language tasks.

By relating to this project, GPT seems to be the most suitable model for this project. Its state-of-the-art performance in several downstream tasks, including text classification and multiple-choice question answering, can be leveraged to understand and extract information from pre-existing user data. Furthermore, its ability to be trained on a large amount of data can be utilized to train the model on a variety ofPDF forms, enabling it to understand and fill these forms accurately. However, the computational cost and memory requirements of GPT should be taken into consideration when deploying this model.

Chapter 3

Software Implementation

3.1 Introduction

3.1.1 Overview

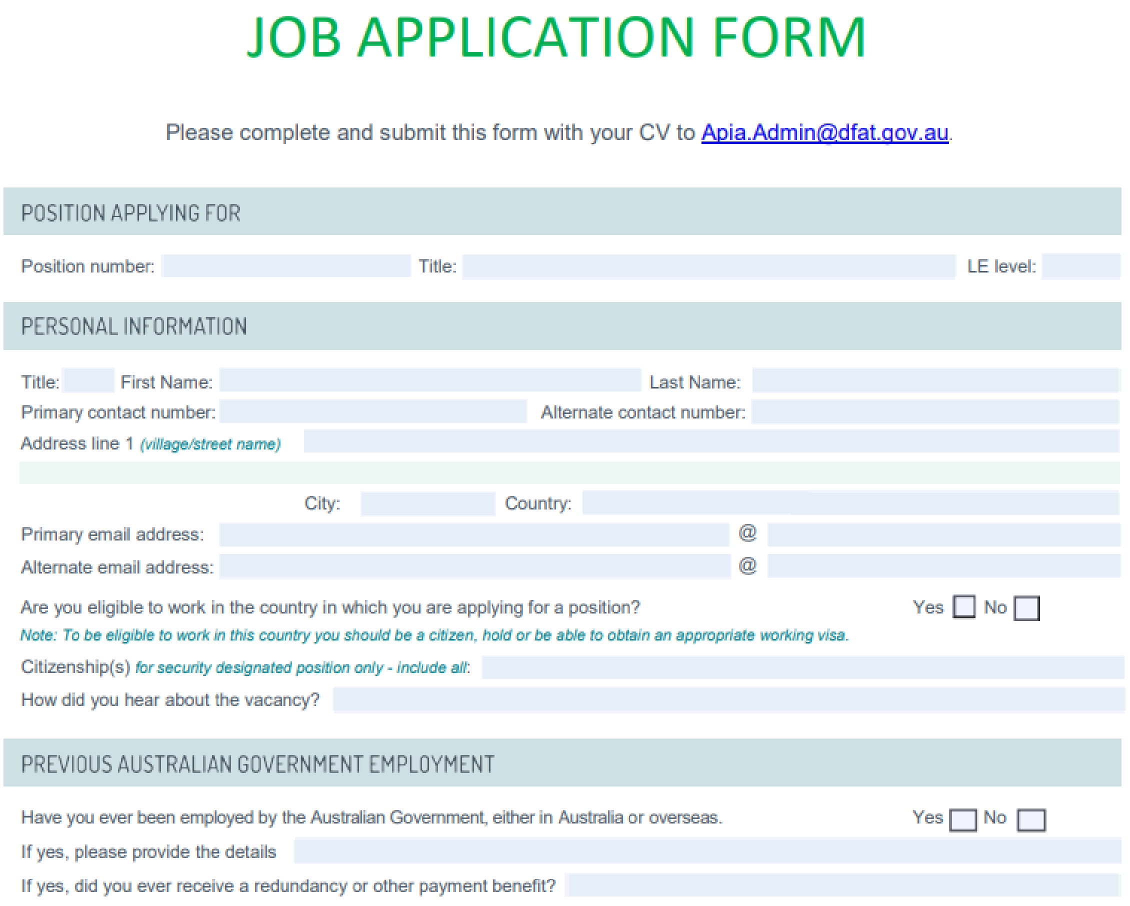
This section mainly introduces the development process of our program, provides some practical use cases, explains the code implementation process, and highlights certain aspects of the program's features.

3.1.2 Software Use Cases

Leveraging automation capabilities, this software simplifies tasks involving repetitive input of personal or historical data in forms, particularly evident in use-cases such as job applications, student registrations, and rental applications. By employing this software, users effectively enhance efficiency and accuracy in these administrative processes. The following is a detailed description of the three use-cases.

Scenario 1: Job Applications

When applying for jobs, applicants often encounter the arduous task of completing numerous application forms that require similar information. For instance, an applicant may apply to several companies, each requiring a unique application form, but the information requested is often similar, like name, address, previous job roles, and educational qualifications. Here, the software becomes highly useful. After the user uploads several completed job application forms (input forms) and a new one that needs to be filled out (output form), the software extracts and matches similar fields from the input forms to populate the output form. If any information is missing, it fills 'N/A' in the field and notifies the user, making the job application process faster and less redundant.



JOB APPLICATION FORM

Please complete and submit this form with your CV to Apia.Admin@dfat.gov.au.

POSITION APPLYING FOR

Position number:		Title:		LE level:	

PERSONAL INFORMATION

Title: First Name: Last Name:

Primary contact number: Alternate contact number:

Address line 1 *(village/street name)*

City: Country:

Primary email address: @

Alternate email address: @

Are you eligible to work in the country in which you are applying for a position? Yes ☐ No ☐

Note: To be eligible to work in this country you should be a citizen, hold or be able to obtain an appropriate working visa.

Citizenship(s) *for security designated position only - include all*:

How did you hear about the vacancy?

PREVIOUS AUSTRALIAN GOVERNMENT EMPLOYMENT

Have you ever been employed by the Australian Government, either in Australia or overseas. Yes ☐ No ☐

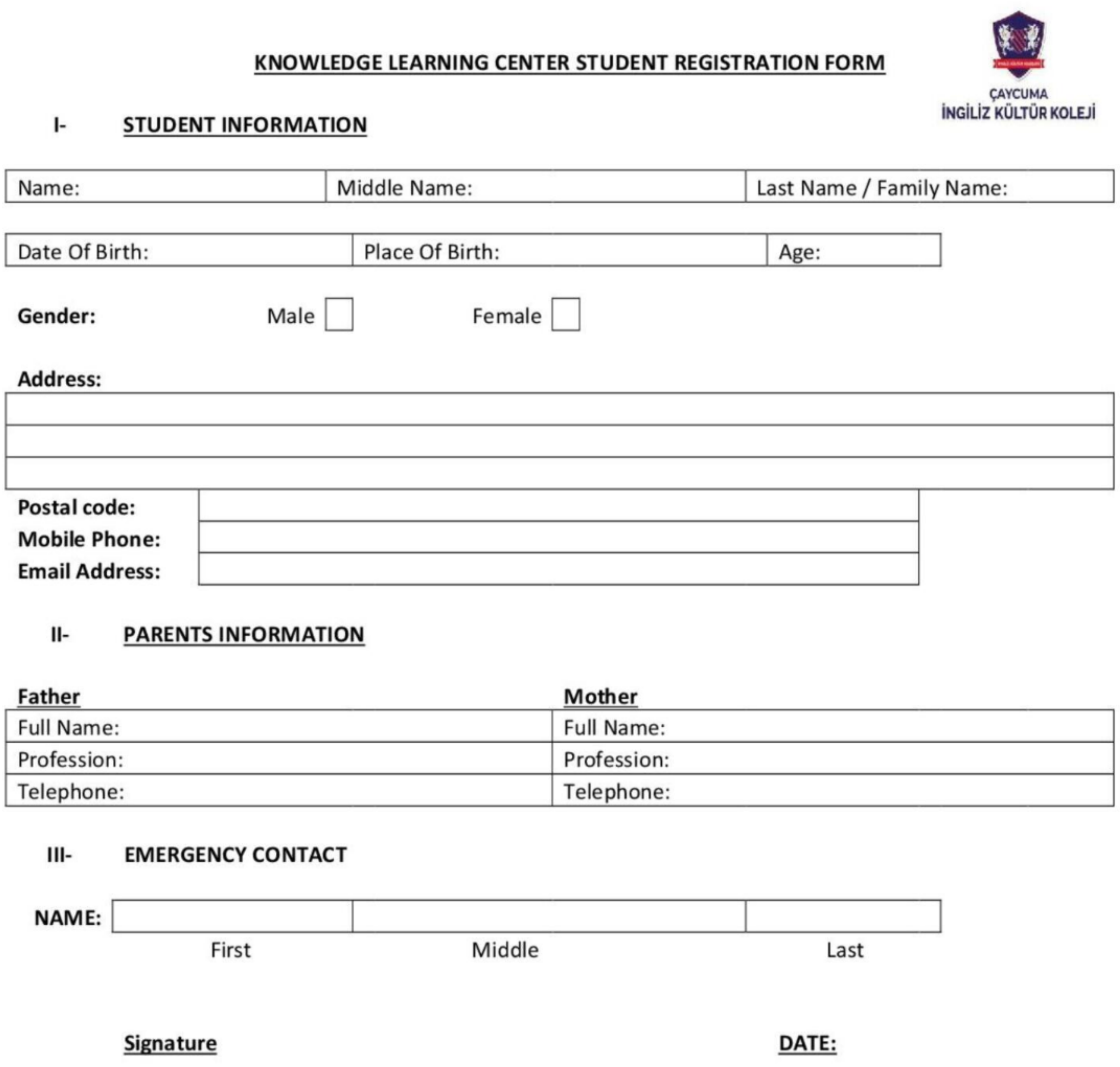
If yes, please provide the details

If yes, did you ever receive a redundancy or other payment benefit?

Figure 3.1: Job Application Form

Scenario 2: Student Registration Forms

Every academic year, educational institutions require students to complete registration forms. These forms typically require standard details like personal information, previous academic details, and emergency contact information. Suppose a student has completed registration forms from previous years (input forms) and now needs to fill out a new one (output form). The software can automatically populate most fields of the output form by comparing it with the input forms. This would save a significant amount of time during the registration process, allowing students to focus more on their academic pursuits rather than administrative tasks.



KNOWLEDGE LEARNING CENTER STUDENT REGISTRATION FORM

ÇAYCUMA
İNGİLİZ KÜLTÜR KOLEJİ

I- STUDENT INFORMATION

Name:	Middle Name:	Last Name / Family Name:

Date Of Birth:	Place Of Birth:	Age:

Gender: Male ☐ Female ☐

Address:

Postal code:
Mobile Phone:
Email Address:

II- PARENTS INFORMATION

Father	**Mother**
Full Name:	Full Name:
Profession:	Profession:
Telephone:	Telephone:

III- EMERGENCY CONTACT

NAME:

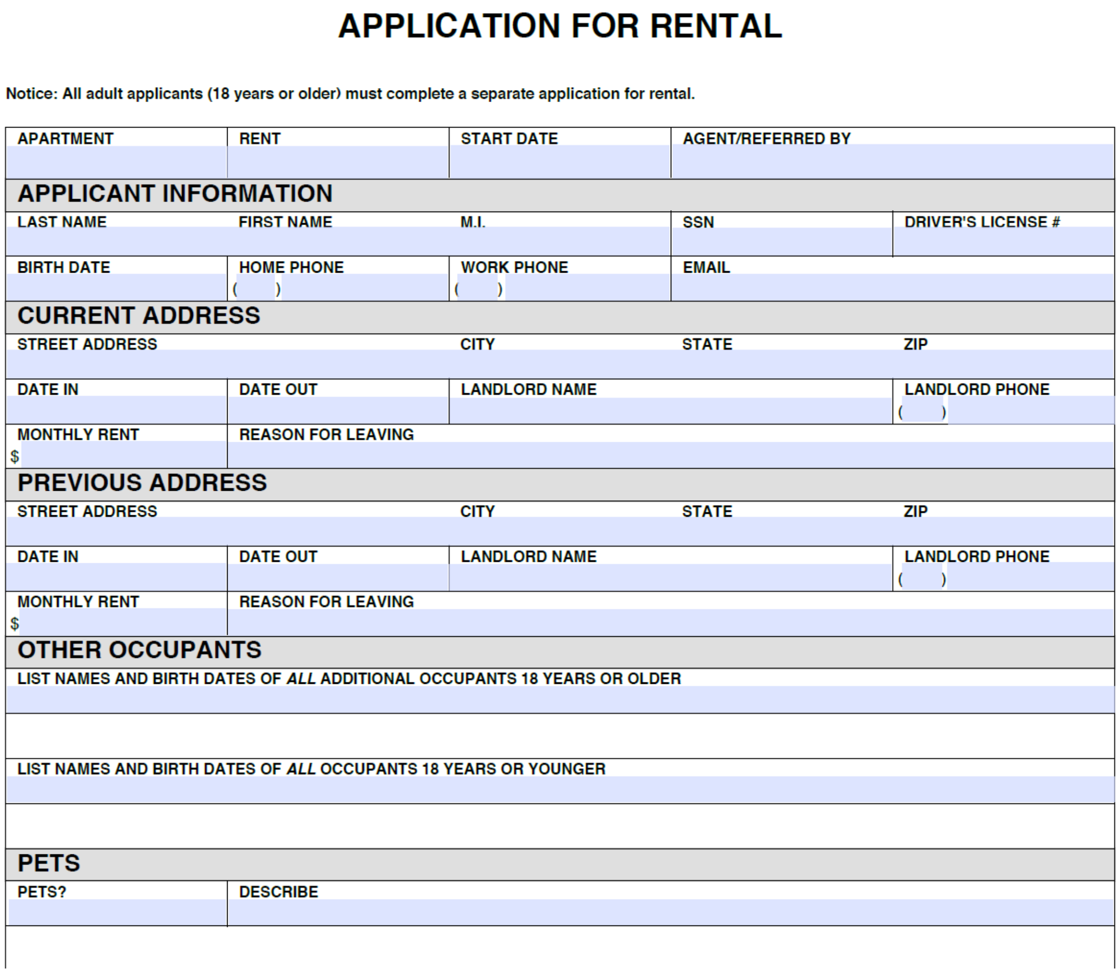
First	Middle	Last

Signature **DATE:**

Figure 3.2: Student Registration Form

Scenario 3: Rental Application Forms

When applying for a rental property, prospective tenants usually have to fill out rental application forms. These forms often require information such as previous addresses, landlord references, and income details. A tenant who is frequently moving may have several completed rental application forms (input forms) and a new one for the next property (output form). By comparing the output form to the input forms, the software can fill out the fields with high accuracy, saving time and effort. In cases where the new form requires unique information (like a new reference), the software will input 'N/A', alerting the tenant to provide this data manually.


APPLICATION FOR RENTAL

Notice: All adult applicants (18 years or older) must complete a separate application for rental.

APARTMENT	RENT	START DATE	AGENT/REFERRED BY	
APPLICANT INFORMATION				
LAST NAME	FIRST NAME	M.I.	SSN	DRIVER'S LICENSE #
BIRTH DATE	HOME PHONE ( )	WORK PHONE ( )	EMAIL	
CURRENT ADDRESS				
STREET ADDRESS		CITY	STATE	ZIP
DATE IN	DATE OUT	LANDLORD NAME		LANDLORD PHONE ( )
MONTHLY RENT $	REASON FOR LEAVING			
PREVIOUS ADDRESS				
STREET ADDRESS		CITY	STATE	ZIP
DATE IN	DATE OUT	LANDLORD NAME		LANDLORD PHONE ( )
MONTHLY RENT $	REASON FOR LEAVING			
OTHER OCCUPANTS				
LIST NAMES AND BIRTH DATES OF *ALL* ADDITIONAL OCCUPANTS 18 YEARS OR OLDER				
LIST NAMES AND BIRTH DATES OF *ALL* OCCUPANTS 18 YEARS OR YOUNGER				
PETS				
PETS?	DESCRIBE			

Figure 3.3: Rental Application Form

3.2 Methodology

3.2.1 Process Overview

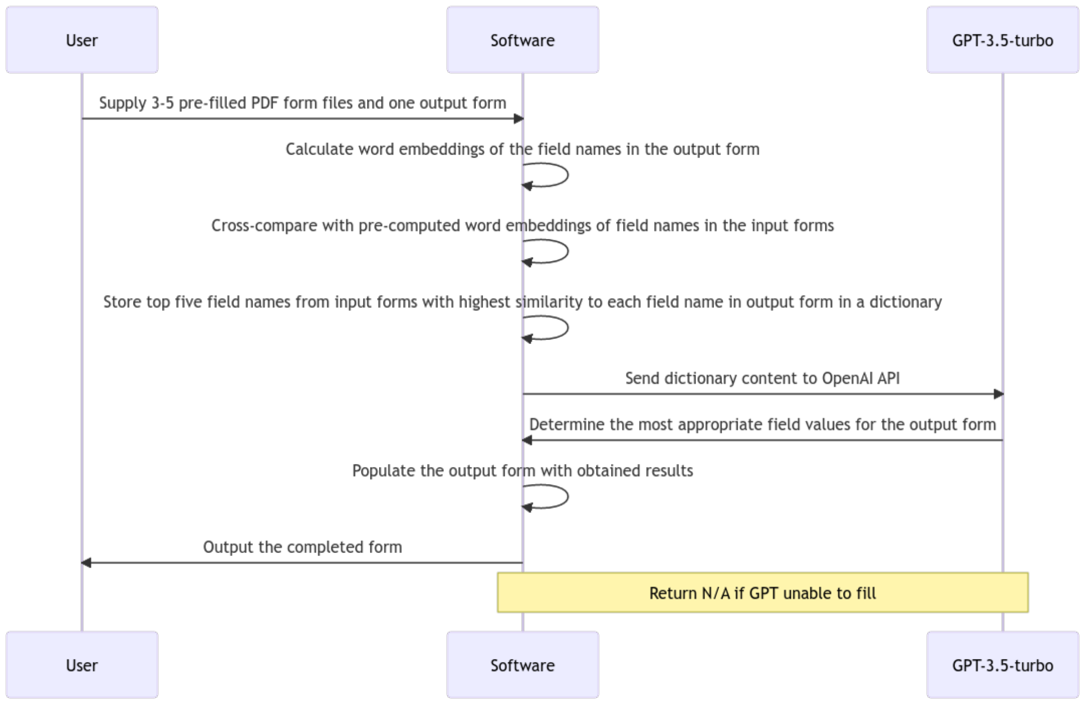

User
Software
GPT-3.5-turbo
Supply 3-5 pre-filled PDF form files and one output form
Calculate word embeddings of the field names in the output form
Cross-compare with pre-computed word embeddings of field names in the input forms
Store top five field names from input forms with highest similarity to each field name in output form in a dictionary
Send dictionary content to OpenAI API
Determine the most appropriate field values for the output form
Populate the output form with obtained results
Output the completed form
Return N/A if GPT unable to fill
User
Software
GPT-3.5-turbo

Figure 3.4: Flowchart for the Process Overview

The following description elaborates on the expected outcome of the software implementation. Firstly, users will supply 3-5 pre-filled PDF form files (referred to as input forms) and one output form that needs to be filled. The software will calculate the word embeddings of the field names in the output form using the GloVe 300D dictionary. These will be cross-compared with the pre-computed word embeddings of field names in the input forms to calculate their cosine similarity.

The top five field names from the input forms that have the highest similarity to each field name in the output form, along with their corresponding field values, will be stored in a dictionary. The software will then read this dictionary and sequentially send its content to the OpenAI API to interact with the GPT-3.5-turbo model. This process will assist in determining the most appropriate field values to fill into the corresponding fields in the output form.

Subsequently, the software will populate the output form with the obtained results and output the completed form. It is important to note that during the interaction with the GPT model, if the model is unable to make a judgment, or if content is missing from the input forms, the system will fill 'N/A' into the relevant fields and notify the user to manually input the answer later. This software aims to offer a systematized approach to form filling and significantly ease the process for the users.

3.2.2 Code Implementation

(1) Importing Libraries: This step involves importing a range of Python libraries that will be crucial for the entire process.

- **openai** is utilized for interacting with the OpenAI API, which allows for the application of the GPT-3.5-turbo model.
- **numpy** is a fundamental library for numerical computation in Python, offering support for large, multi-dimensional arrays and matrices, along with a vast collection of high-level mathematical functions to operate on these arrays.
- **pdfrw** and **pdfminer** are Python libraries specifically designed for reading, writing, and modifying PDF files. They offer low-level access to PDF file structure and content.
- **scipy.spatial.distance** is used to calculate the cosine similarity, a measure that calculates the cosine of the angle between two vectors, which in this case, are the word embedding vectors.
- **gensim** is a Python library for topic modeling and document similarity analysis, which is used here for its word embeddings functionality.

(2) Word Embeddings Loading: GloVe word embeddings are loaded from a predetermined text file. Word embeddings are real-valued vectors, where each word in the corpus is assigned a unique vector. The vectors are generated in a way that words that share common contexts in the corpus are located close to each other in the vector space. In this step, each line from the GloVe file is read, where a line contains a word followed by N numbers (N being 300 in this case), representing the word's vector. Each word and its corresponding vector are stored in **embedding_dict**.

(3) Extract Form Fields: In this step, a function **extract_form_fields** is defined to extract the form fields from a provided PDF file. The function uses the **pdfminer** library to parse the PDF file and locate the form fields within the document structure. It returns a dictionary where each key-value pair corresponds to a form field and its value. If the field is empty, an empty string is assigned as the value.

(4) **Compute Cosine Similarities:** The function **compute_similarity** is implemented in this stage. It calculates the cosine similarity between the given input vector and each vector in the matrix. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The function returns the indices of the vectors that have the highest similarity to the input vector, essentially the topN matches. A threshold can also be set to filter out matches with low similarity scores.

(5) **Match Keys with Values:** The function **match_topn_key_values** is employed to determine the similarity between the keys (i.e., field names) in the input forms and the output form. It first computes the mean of the word embeddings of each word in the key, creating a single 300-dimensional vector representing the key. It then uses the **compute_similarity** function to find the topN matches. If the key does not exist in the word embeddings, it returns None. Otherwise, it returns a dictionary with the topN similar field names from the input forms along with their corresponding field values.

(6) **Constructing a Prompt:** The **cat_prompt** function generates a string that can be used as a prompt for the OpenAI GPT model. The function takes the input key and the corresponding similar keys from the input forms as arguments. It formulates a message that instructs the model to determine the value of the output form field (input key) based on the given input fields (target keys and their values). The prompt includes all target fields and their values and explicitly asks the model to provide only the value to be filled in the output field, returning "N/A" if an inference cannot be made.

(7) **Querying the OpenAI Model:** The **get_openai_result** function sends the constructed prompt to the OpenAI API using the **openai.ChatCompletion.create** function. This function simulates a conversation with the GPT model, where the role "system" sets the model's behavior and the role "user" provides the task. If the rate limit of the OpenAI API is exceeded, it tries again after a delay, up to a maximum of five attempts. The function returns the model's response, which is expected to be the value for the field in the output form.

(8) **Writing Data to PDF:** The function **write_data_to_pdf** fills the fields in the output form. It opens the template PDF file, reads its structure, and locates the form fields. For each field, it uses the **match_topn_key_values** function to find the similar fields in the input forms. If no matches are found or if there's only one match (with the exact same field name), it sets the value to the matched value or leaves it empty. Otherwise, it constructs a prompt using **cat_prompt**, sends it to the OpenAI API using **get_openai_result**, and fills the field with the model's response.

(9) Process Multiple PDFs: An outer loop is implemented to extract the form fields and their values from multiple input PDF files. The results are combined into a single dictionary, with the field names as keys and the field values as values.

(10) Compute Word Embeddings for PDF Keys: Compute Word Embeddings for PDF Keys: For each key in the combined results, it computes the average of the word embeddings of the words in the key, yielding a single 300-dimensional vector representing each field name. This is stored in **pdf_key_vecs**, with the corresponding keys in **pdf_keys**.

(11) Complete the Output Form: Finally, the **write_data_to_pdf** function is called with the path of the output form and the desired path of the completed form. The function reads the output form, locates the form fields, matches each field with the similar fields in the input forms, queries the OpenAI model if necessary, and fills the fields with the obtained values. The completed form is then saved to the specified path.

3.3 Key Features of Implementation

3.3.1 Precomputation for Word Embeddings

A key feature of this software lies in the strategy of precomputing word embeddings for all field names and values in the input forms. By calculating these embeddings in advance, the software improves its performance by reducing the computational time during the form-filling process. Through precomputing these embeddings, the software shifts the computationally demanding task from the form-filling process to an earlier phase, enhancing overall system performance.

This strategy represents a computational trade-off, as the storage of precomputed embeddings eliminates the need for repetitive calculations for each form-filling task. By implementing this approach, the software ensures that its performance doesn't degrade during the form-filling process, enhancing its reliability and responsiveness.

The precomputation strategy becomes increasingly valuable when handling a large number of forms or complex forms. It keeps the system's performance steady regardless of the volume or complexity of the forms, providing consistent and efficient user experience. In conclusion, the precomputation of word embeddings exemplifies a thoughtful design choice, improving the software's capability to handle automated form completion tasks.

3.3.2 Filter Conditions

The second notable aspect of the software implementation lies in the introduction of filtering conditions during the computation of cosine similarity between field names in input and output forms. These filtering conditions serve to optimize the efficiency of the process and manage resource usage effectively.

In particular, when the cosine similarity score equals 1.0, signifying identical field names in both the input and output forms, the software takes advantage of this perfect match. It simply transfers the field values from the input forms to the corresponding fields in the output form, obviating the need for further processing.

Contrarily, when the cosine similarity is less than 0.5, indicating a high degree of difference between the field names in the input and output forms, the software directly inputs 'N/A' in the relevant field of the output form. This decision mitigates the risk of inaccurate form filling and prevents unnecessary usage of computational resources.

By establishing these two conditions, the software avoids unnecessary querying to GPT in situations where the outcome is either a guaranteed match or highly unlikely. The result is a considerable savings in computational time and a reduction in the number of tokens sent to GPT, thereby contributing to cost-efficiency. Through this prudent use of filtering conditions, the software ensures a more efficient and cost-effective approach to automated form filling.

3.3.3 Model Selection

The decision regarding the model selection for automated form-filling tasks is a delicate balance between performance and cost efficiency. In a detailed study conducted by (Ye et al., 2023), it was discerned that GPT-3.5 Turbo parallels the performance of text-davinci-003 across most tasks, despite minor underperformance in Machine Reading Comprehension (MRC), Part-of-Speech (POS), and Relation Extraction (RE) tasks. Notwithstanding this slight compromise in performance, the notable cost advantage of GPT-3.5 Turbo, which is only one-tenth that of text-davinci-003, makes it a more economically viable choice.

Further, (Ye et al., 2023) also discovered an improvement in model performance with an increased number of prompts. However, it's intriguing to note that GPT-3.5 Turbo, despite its progressive features, demonstrated less robustness in Natural Language Inference (NLI) tasks compared to its predecessors. This suggests that when using GPT-3.5 Turbo for automated PDF form filling, there may be a need for additional prompts or predefined information. Therefore, owing to its cost-effective performance, GPT-3.5 Turbo seems to be a promising choice for automated PDF form filling.

3.3.4 Retry Logic

Another feature of the system is the retry logic. When interacting with GPT, stability is paramount given the frequent occurrences of network errors and other server-related issues, which could potentially cause interruptions to the entire operation. To mitigate this, we've implemented a retry logic mechanism, specifically designed to enhance resilience against unforeseen server errors.

The logic operates on a principle of graduated retries. It comes into effect when a server error is detected during the interaction with the GPT. Upon encountering such a problem, the system is configured to initiate a sequence of retries, with the number of attempts capped at five.

Notably, this system doesn't merely repeat the requests in quick succession. Instead, it incorporates an essential element of progressive back-off. Each subsequent retry comes with a gradually increasing wait time before the next attempt. This tactic provides a temporal buffer, allowing potential issues within the server to resolve and increasing the likelihood of a successful communication in the next attempt.

However, should the system continue to receive errors despite exhausting the five retries, it resorts to returning an error message to the user. This provision ensures the process does not hang indefinitely while waiting for server responses, thereby improving the overall reliability and user experience of the system. Through this strategic approach, we're able to significantly increase the stability of interactions with GPT.

3.3.5 Prompt Refinement

Throughout the development of this project, a crucial aspect of focus was the refinement of the prompts sent to the GPT model. This process constituted an iterative cycle of trial and evaluation, with a series of adjustments being made to optimize the balance between the token count sent to GPT, the accuracy of the response received, and the processing time.

In the earliest stages of the development, the prompts were more verbose, often including a detailed instruction set for the GPT, such as deducing and filling in specific information based on input, stipulating the return of 'None' if the deduction was impossible, and providing a specific format for the result. However, these lengthy prompts increased both processing time and token usage, and thus necessitated further optimization.

A step towards efficiency was the removal of the instructions about returning the 'closest' value if deduction was uncertain. By focusing GPT's attention strictly on exact matching or the lack thereof, the model's response quality was improved, and the processing time was reduced.

The prompt refinement process further evolved to include instructions regarding handling blank or uncertain answers, with a shift towards returning 'None' under these conditions. This adjustment guided the GPT towards producing more meaningful and accurate responses.

In the final stage, the prompt was pared down to its most essential form: requesting the model to deduce and fill in the output field based on the input, and to return 'N/A' if the inference wasn't possible, providing only the answer without any explanation.

This iterative process of prompt refinement ensured an optimized balance among token usage, response accuracy, and processing time, contributing to the efficiency and efficacy of the system.

Chapter 4

Experimental Results

4.1 Introduction

This chapter aims to illustrate and discuss the experimental results of our program in different scenarios. It encompasses the methodological planning for the experiment, including the preparation of testing materials, establishment of evaluation metrics, and the structured assessment of the program's functionality and performance. The program's accuracy, responsiveness, and adaptability are under evaluation in two scenarios: job applications and rental applications. This chapter presents the results, discussions, strengths and weaknesses of the program to offer a comprehensive understanding of its capabilities. It also suggests potential areas for improvement.

4.2 Preparation

In order to ensure that the program can successfully pass functional testing and performance testing, we first need to design and create testing materials and specify evaluation metrics.

4.2.1 Testing Materials

In preparation for the evaluation of our developed algorithm, we meticulously generated a range of testing materials pertinent to two distinct scenarios: Job Applications and Rental Applications. To ensure a comprehensive and unbiased assessment of our algorithm's performance, each scenario comprised of three pre-populated Input forms along with a blank Output form. In this section, I will elucidate the specifics of these testing materials and explain how their design feeds into the broader context of our study.

For both scenarios, it is important to note that we deliberately selected standardized forms with accurately labeled fields. The purpose of doing this is to enable the program to accurately identify field information in order to match content accurately and ensure smooth program presentation.

Firstly, we will showcase the design process of job applications. These forms were filled with a wealth of detailed personal information such as the applicant's name, address, phone number, and email. Additionally, recognizing the distinctive requirements of this scenario, we incorporated sections for education background, professional experience, and references.

What's especially noteworthy is our effort to test the adaptability of our program by diversifying the expression of the applicant's name. Three different formats were introduced to achieve the desired outcome: Given Name/First Name, Surname/Last Name, and Full Name. This approach serves two primary objectives. Firstly, it reflects real-world situations with distinct variations in data presentations. Secondly, it enables assessment of program response and adaptability to diverse contexts.

As for the second scenario, rental applications, we followed a similar protocol with regard to the personal information section. However, to cater to the specific needs of this scenario, we also included details about the applicant's cohabitants, previous landlord, former rental address, and pet information. The inclusion of these categories not only improves the ecological validity of our testing materials but also gauges the program's accuracy when confronted with similar types of information.

In addition to the standard personal information, the Rental Applications also contained particulars about expected rent, rental history, employer details, and banking information. This adds another layer of complexity for our program to navigate and thus, makes the evaluation more rigorous.

In essence, we believe that both Job Applications and Rental Applications serve as representative scenarios that embody the diversity and complexity of the practical world. Should our program successfully pass the Functionality and Performance Evaluation in these scenarios, it would validate its practical applicability. By ensuring a robust preparation phase, we strive to provide a thorough and balanced assessment of the program's capacity to operate in diverse, real-world settings.

4.2.2 Evaluation Metrics

In the bid to rigorously evaluate both the functionality and performance of our program, we employed three crucial evaluation metrics: Accuracy, Responsiveness, and

Adaptability. Each of these metrics addresses a different facet of the program's efficacy, ensuring a comprehensive and thorough assessment of its performance.

Accuracy

Accuracy is aimed at gauging the precision with which the program populates data. For the accurate calculation of this metric, we initiated a meticulous process of collecting all filled data from the three input forms. Leveraging this collated data, we manually filled the output forms, hereinafter referred to as 'example_out'. Any fields that couldn't be populated were labeled with an 'N/A' tag. Subsequently, we employed a Python script to compare the field values between the output forms and the 'example_out' PDFs. We computed Accuracy using the formula:

Accuracy = (Number of Matching Fields between Output Forms and 'example_out' PDFs) / (Total Number of Fields with Values in 'example_out')

By adopting this process, we ensured an objective measure of the accuracy of the data filled in by the program.

Responsiveness

To measure the responsiveness of the program, we leveraged Python's built-in 'time' module. Following each successful run of the program, we recorded three different time parameters: the total runtime of the program, the model loading time, and the time taken to write into the PDF (inclusive of the communication time with GPT). After running this protocol for ten instances, we computed the average time for each of these parameters. This average served as a 'ground truth' time reference for subsequent runs.

The rationale behind this approach lies in the varying communication time with GPT. Only after recording data over multiple runs can we arrive at a representative 'total runtime' that can serve as a reliable reference for comparative evaluation. Thus, our measure of responsiveness goes beyond mere execution speed, incorporating a more nuanced understanding of system performance.

Adaptability

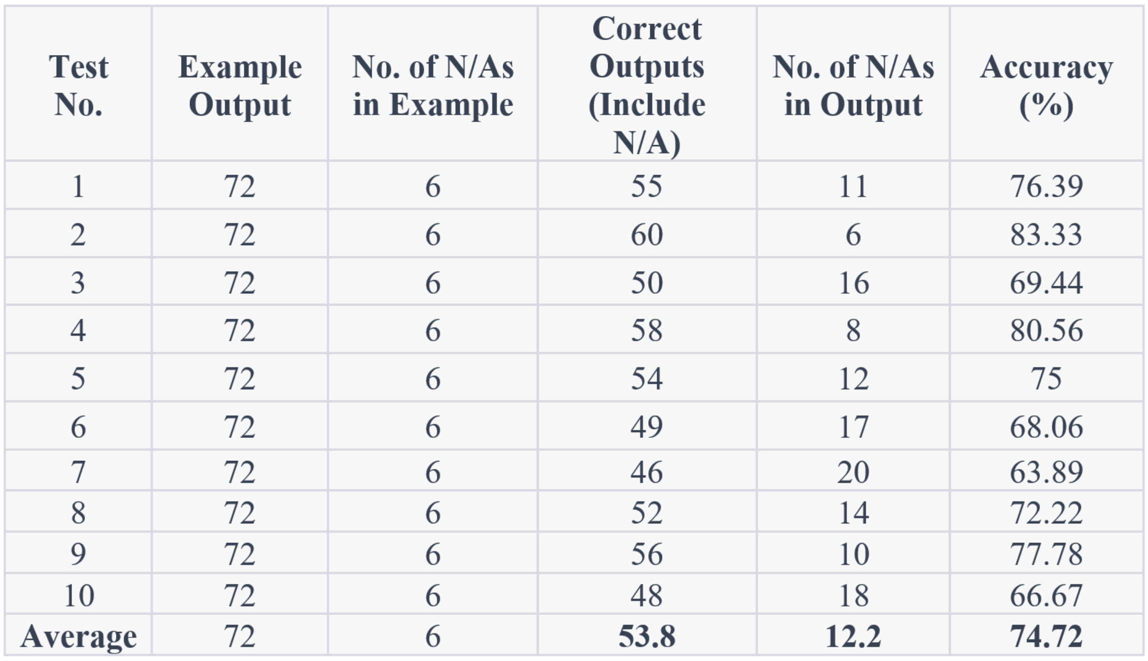
Testing for adaptability has already been considered while preparing the testing materials. As mentioned earlier, we prepared two different scenarios for testing: Job Applications and Rental Applications. Within these scenarios, we intentionally infused diversity into the content of the forms.

In the Job Applications, for instance, we used three different ways to denote names: Given name/First Name, Surname/Last Name, and Full Name. This diversity aimed to assess how the program navigates varied expressions of the same information type. Similarly, in the Rental Applications, we included fields such as cohabitants, previous landlord, previous rental address, and pet information. Although these fields contain similar information, they are inherently different, thereby testing the program's discernment capabilities.

By employing these three metrics, we created a multi-dimensional evaluation framework that is well-equipped to test the program's accuracy, responsiveness, and adaptability. By placing equal emphasis on each of these aspects, we can ensure a comprehensive evaluation of the program's overall functionality and performance. This robust approach contributes to a more nuanced understanding of the strengths and potential areas for improvement in our system.

4.3 Functionality and Performance Evaluation

4.3.1 Accuracy



Test No.	Example Output	No. of N/As in Example	Correct Outputs (Include N/A)	No. of N/As in Output	Accuracy (%)
1	72	6	55	11	76.39
2	72	6	60	6	83.33
3	72	6	50	16	69.44
4	72	6	58	8	80.56
5	72	6	54	12	75
6	72	6	49	17	68.06
7	72	6	46	20	63.89
8	72	6	52	14	72.22
9	72	6	56	10	77.78
10	72	6	48	18	66.67
Average	72	6	**53.8**	**12.2**	**74.72**

Figure 4.1: Accuracy Results for Rental Applications.

The table describes the results of our accuracy tests. Here is a further description of the table:

Test No: stands for the identification number of each test conducted.

Example Output: is a static value, representing the total number of fields in the example forms. It stands at 72 in all the tests.

No. of N/As in Example: is also a static value, indicating the number of fields marked as N/A in the example forms, with a count of 6 in all tests.

Correct Outputs (Include N/A): is a calculated value representing the number of fields that have been accurately filled in the output forms, which also includes the correct recognition of N/A fields.

No. of N/As in Output: represents the number of fields marked as N/A in each test's output forms. This value fluctuates from test to test.

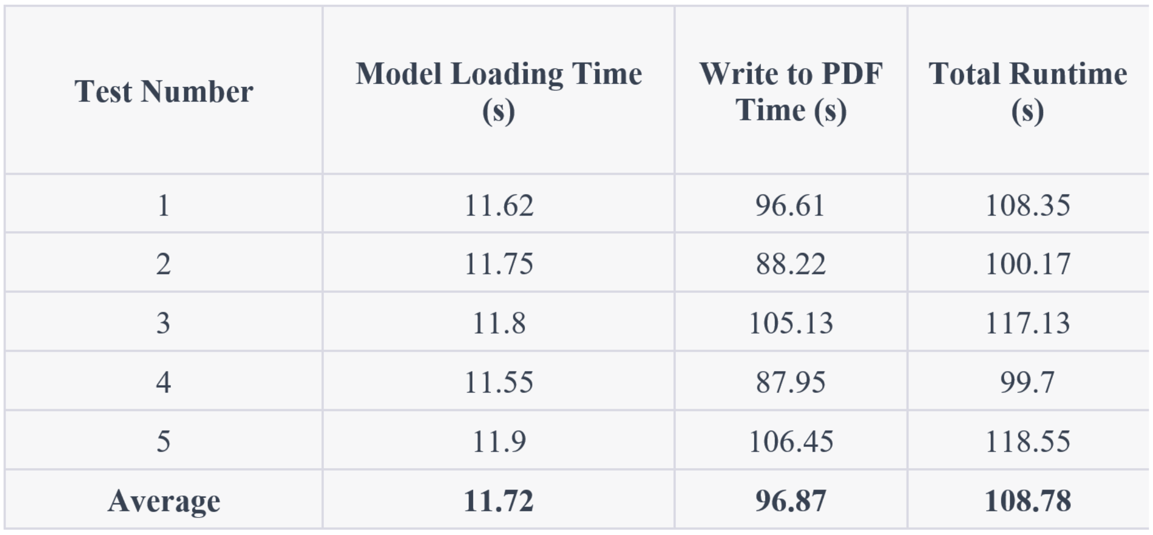
Accuracy (%): is the result of the ratio of "Correct Outputs" to the "Example Output," multiplied by 100 to give a percentage. This percentage depicts the accuracy level of each test.

Analyzing the data, the average "Correct Outputs (Include N/A)" stands at 53.8, and the average "No. of N/As in Output" is 12.2. The mean value of accuracy is 74.72. These values are averaged over the ten tests conducted.

After examining the data, some significant findings emerged. The data showed that the system frequently returned "N/A" at a higher rate than in the example. This could be due to certain prompts, and it's possible that modifying them might help solve this issue. According to the data, system accuracy stands at 74.72%. Additionally, there exists a negative relationship between "N/A" responses and system accuracy: as the number of "N/A" responses increases, accuracy decreases. Therefore, improving prompts would significantly enhance precision.

Additionally, many errors were discovered to stem from content being incorrectly positioned. For instance, in some outputs, the phone numbers were fragmented and filled in the wrong order. In the typical "xxx-xxxx-xxx" format, the system filled the entire phone number into the first segment, resulting in an incorrect format like "1234567890-N/A-N/A".

Therefore, considering the system's average accuracy of 74.2% when it includes "N/A", the system can be deemed to be in a usable state. Further functional testing can enhance this performance, paving the way for optimization and more accurate results.



Test Number	Model Loading Time (s)	Write to PDF Time (s)	Total Runtime (s)
1	11.62	96.61	108.35
2	11.75	88.22	100.17
3	11.8	105.13	117.13
4	11.55	87.95	99.7
5	11.9	106.45	118.55
Average	**11.72**	**96.87**	**108.78**

Figure 4.2: Responsiveness Results for Rental Applications.

4.3.2 Responsiveness

Exploring the next part of our evaluation, we focus on the system's responsiveness through several parameters - the Model Loading Time, Write to PDF Time, and Total Runtime - for each test iteration.

Firstly, the "Test Number" offers an identification system for each separate test iteration. We then examine the "Model Loading Time", which measures, in seconds, the duration it takes for the system to load the model. A crucial observation is the stability of these loading times across all tests, which hover around an average of 11.72 seconds. This uniformity in model loading time provides a consistent baseline for evaluating the system's performance.

Next, we turn to the "Write to PDF Time". This indicator measures the duration necessary for the system to transcribe the output into a PDF format. This timeframe is a bit more varied compared to the model loading time, primarily due to the incorporation of the time required to communicate with the GPT. Despite these variations, the mean time of about 96.87 seconds remains within a reasonable range.

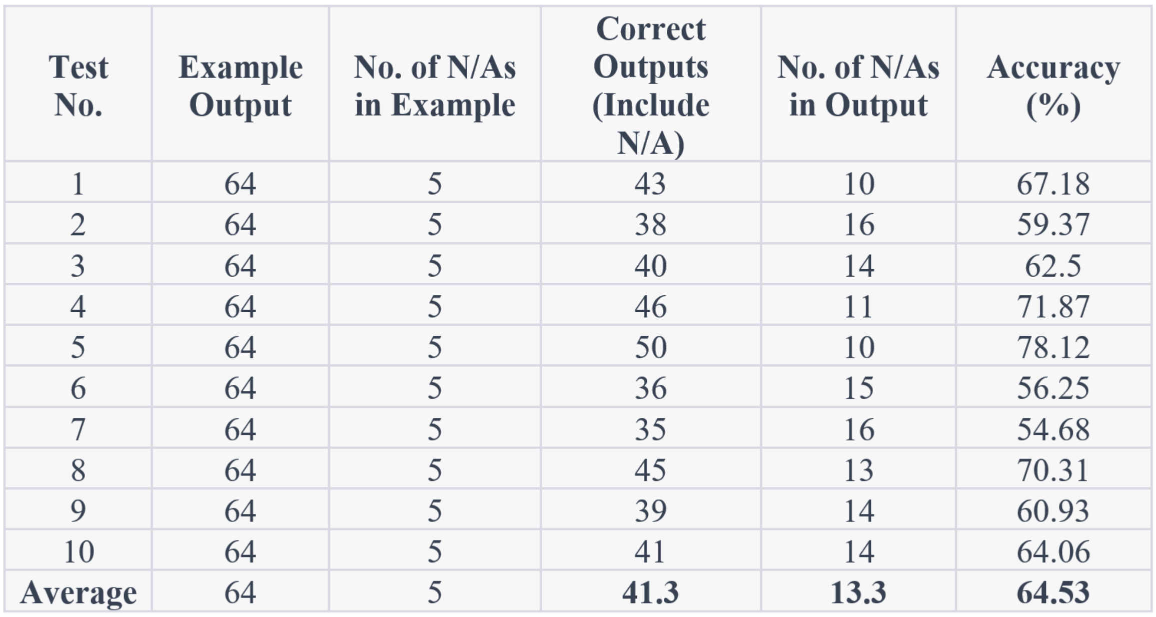
Finally, the "Total Runtime" combines the previous two parameters, providing a comprehensive measure of the system's overall efficiency from model loading to PDF generation. The average total runtime comes to approximately 108.78 seconds.

By carefully scrutinizing these metrics, we gain insights into the system's operational efficiency. The constant and efficient model loading time combined with the overall acceptable total runtime suggests a level of responsiveness that should meet user expectations. Though the "Write to PDF Time" shows some inconsistencies due to the interaction with GPT, it remains within a bearable limit, ensuring the system's feasibility for real-world application.

Despite the observed fluctuations in the Write to PDF Time, the system showcases commendable performance regarding responsiveness. All processes stay within the two-minute mark on average, indicating a balanced and efficient performance that does not compromise user experience. This analysis affirms that the system's level of responsiveness is robust and fit for practical use.

4.3.3 Adaptability

Adaptability in a system is evaluated by its performance across different use cases. In our context, it pertains to the proficiency of the program to accurately fill out job application forms while maintaining efficient operation times. Our investigation of the program's adaptability has been meticulously conducted through the assessment of two significant metrics: accuracy and responsiveness.

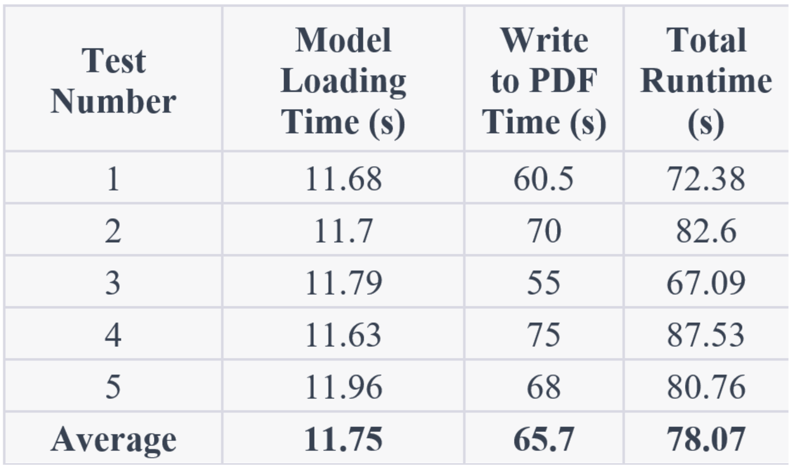


Test No.	Example Output	No. of N/As in Example	Correct Outputs (Include N/A)	No. of N/As in Output	Accuracy (%)
1	64	5	43	10	67.18
2	64	5	38	16	59.37
3	64	5	40	14	62.5
4	64	5	46	11	71.87
5	64	5	50	10	78.12
6	64	5	36	15	56.25
7	64	5	35	16	54.68
8	64	5	45	13	70.31
9	64	5	39	14	60.93
10	64	5	41	14	64.06
Average	64	5	**41.3**	**13.3**	**64.53**

Figure 4.3: Accuracy Results for Job Applications.

The average accuracy for job application forms was found to be 64.53%. In comparison to the rental scenario, this demonstrates a decline of nearly 10%. Upon closely monitoring the operational nuances of the program, we discovered that this decrease in accuracy might stem from the ambiguity in the field names pertaining to academic history in the output forms of this particular scenario. It was noted that the Generalized Pre-training Transformer (GPT) struggled to comprehend and correctly populate these fields due to their vague descriptors, leaving an N/A instead. This underscores a limitation in the adaptability of our program to less than ideally formatted or indistinct field names.

Despite this shortcoming, we believe that the issue can be mitigated significantly by employing well-structured forms for testing. This potential improvement emphasizes the flexibility and adaptability of the system to conform to different standards of input forms. It also highlights the ongoing learning and improving nature of the system, showing promise for better performance in subsequent trials.



Test Number	Model Loading Time (s)	Write to PDF Time (s)	Total Runtime (s)
1	11.68	60.5	72.38
2	11.7	70	82.6
3	11.79	55	67.09
4	11.63	75	87.53
5	11.96	68	80.76
Average	**11.75**	**65.7**	**78.07**

Figure 4.4: Responsiveness Results for Job Applications.

Moving to the responsiveness of the system, there is a noteworthy reduction in the total runtime, primarily attributable to the decreased amount of data required to be filled in the job application forms. Yet, the model loading time has remained relatively stable, reinforcing our understanding that the majority of the time consumed by the program is in communicating with the GPT.

In conclusion, while there is a certain reduction in accuracy when transitioning from the rental to the job application scenario, the system appropriately decreases the total operation time in response. Furthermore, the program smartly places an N/A marker in instances where a definitive output could not be determined, allowing users to review and edit after the system run. In light of these observations, the overall performance of the program in terms of adaptability can be considered satisfactory. It balances precision with time efficiency and provides user-friendly feedback for uncertain outputs, thus demonstrating reasonable adaptive capability.

Chapter 5

Limitations

This program has limitations in specific areas, which will be discussed in two sections: software incapabilities and general limitations.

5.1 Software Incapabilities

The first significant limitation arises from the program's inability to effectively handle inaccurately named field labels in PDF forms. While collecting our testing materials, we noticed that a substantial number of non-standardized PDF forms do not properly label each field. Many forms use generic labels like "default_1" or "default_X". However, our program depends on Natural Language Processing (NLP) to accurately interpret and comprehend field names. Consequently, the program cannot process necessary information when it faces improperly or generically labeled fields. In these circumstances, the program defaults to marking all outputs as "N/A". This limitation represents a substantial barrier to the program's functionality, particularly when dealing with poorly structured or non-standardized forms.

The second major limitation lies in the inherent design of our software. Our program is specifically built to interact with 'fillable' or interactive PDF forms. This feature enables the software to read, interpret, and populate fields within these types of forms. However, for PDF forms that lack interactive fields, the program faces a fundamental issue. Although the program can read and interpret the form's content, it cannot interact with the form or populate it with information. This effectively renders the program ineffective in dealing with non-interactive forms, significantly limiting its overall usefulness and applicability.

5.2 General Limitations

Moving beyond software-specific issues, there are certain overarching limitations that our project faces.

One significant area is the design of our evaluation metrics for testing adaptability. The current structure may not encompass all the critical aspects required to adequately measure the program's ability to cope with varied scenarios. The team faced limitations with creating a comprehensive set of PDF form scenarios for testing, due to time constraints. Consequently, the data used for testing didn't encompass all potential real-world applications' use-cases. Therefore, to better evaluate software performance, the team suggests increasing the number of test scenarios to at least five. Each scenario should comprise 3-5 different output forms. This would allow for a more comprehensive and representative evaluation of the software's versatility.

Another concern is the software's high dependency on the GPT model for its output. The accuracy of the GPT-3.5 model, which is limited, often results in varied answers to the same question. This variation, in turn, creates an inconsistency in the program's output if it is run multiple times. We have experimented with the more advanced GPT-4 model, which does offer some improvements in accuracy. However, the cost considerations linked with using this advanced model led us to continue with the GPT-3.5-turbo model.

Chapter 6

Future Development

For the future development plans of the program, we will describe them from two aspects: performance enhancement and ethical issues.

6.1 Performance Enhancement

Improving the functional performance and metrics of our program can be approached from the perspectives of Accuracy and Responsiveness.

For Accuracy, we can enhance our application by continually optimizing and experimenting with different prompts, finding the sweet spot between efficiency and precision. Recent announcements from (OpenAI, 2023) suggest a significant cost reduction for the GPT-4 model API, allowing us to employ this advanced model to bolster our program's accuracy.

Meanwhile, the program's responsiveness is affected by the loading time of the NLP model and the time spent interacting with the GPT. At present, the model begins to load every time the program launches, and this repetition negatively impacts responsiveness. To rectify this, future development could implement a system where the program preloads the model, allowing the user to leverage the loaded model immediately upon importing a table. Once the user's session ends, instead of terminating the program, it will enter a standby mode, ready for the next user request.

The interaction time with the GPT model, governed primarily by network factors, also has room for improvement. The current interaction with the GPT model, facilitated through a network API, presents challenges due to unstable network conditions. Despite having a Retry Logic in place to minimize interruptions, significant time wastage can still occur during the retry process. An alternative approach could be the utilization of localized models like LLaMa (Ye et al., 2023) or Alpaca (Wu et al., 2023). However, as these models might not achieve the same level of accuracy as GPT, we would need to further refine the logic of using these models.

6.2 Ethical Issues

Finally, we must take into account ethical issues pertaining to data privacy, particularly given the fact that our application processes user-supplied table inputs, which may contain sensitive or private data. As the GPT model is a third-party API and we have no control over how OpenAI handles the data we send, privacy protection is a fundamental concern. To ensure full data protection, we could contemplate the usage of offline language models, such as LLaMa (Ye et al., 2023) or Alpaca (Wu et al., 2023), which have been designed with privacy in mind.

In addition, it is crucial to establish strict data retention policies. We need to guarantee that after each session, any files generated during the process that might contain user information are thoroughly deleted. This aligns with the principles of data minimization and storage limitation, as outlined in several privacy frameworks (Greenleaf, 2021). However, the specifics of these policies should be tailored to comply with relevant legal and regulatory requirements, which may vary significantly by jurisdiction.

Chapter 7

Conclusions

In this research, we have delved into vector semantics methods including Word2Vec, LSA, and GloVe, existing form filling methods, and various large language models (LLMs) such as GPT, LLaMa, and Alpaca. A primary focus of this study was the application of GPT for automating PDF form filling processes, whose potential was substantiated through a comprehensive software implementation.

Our methodology harnessed GloVe 300D to compute word embeddings of field names and calculated cosine similarities for accurate matching. The resultant dictionary was parsed through GPT-3.5-turbo, guiding the system to output the most suitable field values. The software implementation boasted key features like pre-calculation of cosine similarity for speed, cost-effective use of GPT, and a robust retry logic.

The experimental results demonstrated considerable success of our approach. The system performance was evaluated through metrics of accuracy, responsiveness, and adaptability. Though certain limitations were observed, such as software incapability in certain areas and general restrictions, these offer avenues for future enhancements.

Prospective improvements encompass refining the functionality and performance indicators, chiefly the accuracy and responsiveness of the system. Moreover, the ethical implications that may arise with more widespread and sophisticated use of this technology cannot be overlooked.

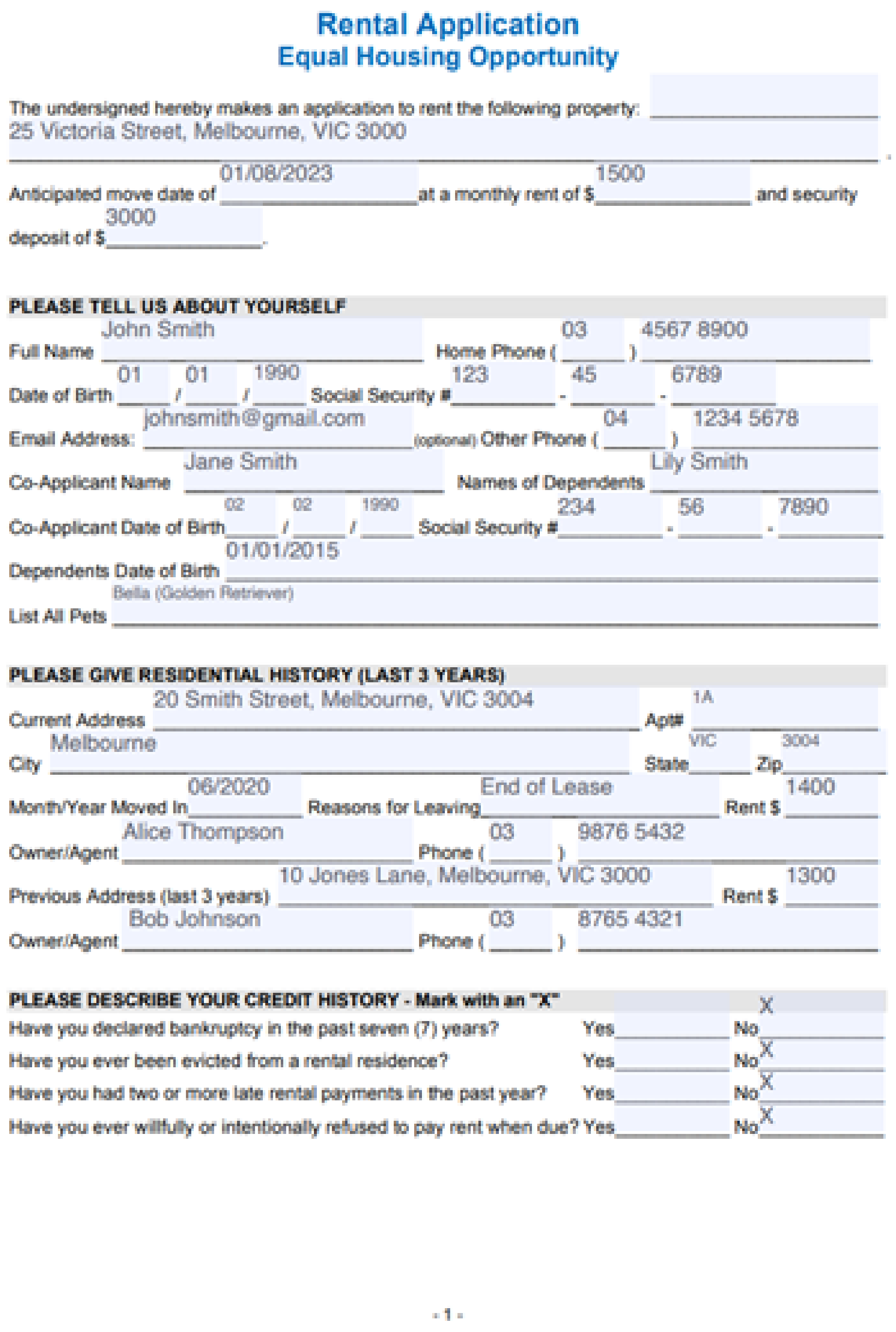
Overall, this research illuminated the potential of combining traditional NLP techniques and LLMs, especially the GPT series. With advancements in AI and machine learning, such applications are bound to bring a significant shift in how data is handled, improving efficiency, accuracy, and user experience. However, the journey ahead still requires careful navigation, with a vigilant eye on the inherent limitations and ethical aspects. As the technology evolves, the potential for positive impact continues to grow, enabling numerous possibilities for practical and innovative applications.

Appendix A

A.1 Python Codes to Calculate Accuracy

```
import io
from pdfminer.converter import TextConverter
from pdfminer.pdfinterp import PDFPageInterpreter, PDFResourceManager
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
from pdfrw import PdfReader


def read_pdf(pdf_file):
    resource_manager = PDFResourceManager()
    fake_file_handle = io.StringIO()
    converter = TextConverter(resource_manager, fake_file_handle,
                                   laparams=LAParams())
    page_interpreter = PDFPageInterpreter(resource_manager, converter)

    with open(pdf_file, 'rb') as fh:
        for page in PDFPage.get_pages(fh,
                                      caching=True,
                                      check_extractable=True):
            page_interpreter.process_page(page)

        text = fake_file_handle.getvalue()

    # close open handles
    converter.close()
    fake_file_handle.close()

    return text


def read_pdf_fields(pdf_file):
    pdf = PdfReader(pdf_file)
    fields = pdf.Info
    return fields


def compare_pdfs(pdf_file1, pdf_file2):
    text1 = read_pdf(pdf_file1)
    text2 = read_pdf(pdf_file2)
    fields1 = read_pdf_fields(pdf_file1)
    fields2 = read_pdf_fields(pdf_file2)

    total_fields = sum(1 for v in fields1.values() if v != '')
    common_fields = sum(1 for k, v in fields1.items() if fields2.get(k
                                       ) == v and v != '')

    accuracy = common_fields / total_fields if total_fields else 0

    return accuracy

pdf_file1 = 'example_form1.pdf'
pdf_file2 = 'Output_form_result.pdf'

print(compare_pdfs(pdf_file1, pdf_file2))
```

A.2 Three Input Rental Applications



Rental Application
Equal Housing Opportunity

The undersigned hereby makes an application to rent the following property: 25 Victoria Street, Melbourne, VIC 3000.

Anticipated move date of 01/08/2023 at a monthly rent of $1500 and security deposit of $3000.

PLEASE TELL US ABOUT YOURSELF

Full Name	John Smith
Home Phone	(03) 4567 8900
Date of Birth	01 / 01 / 1990
Social Security #	123 - 45 - 6789
Email Address:	johnsmith@gmail.com
(optional) Other Phone	(04) 1234 5678
Co-Applicant Name	Jane Smith
Names of Dependents	Lily Smith
Co-Applicant Date of Birth	02 / 02 / 1990
Social Security #	234 - 56 - 7890
Dependents Date of Birth	01/01/2015
List All Pets	Bella (Golden Retriever)

PLEASE GIVE RESIDENTIAL HISTORY (LAST 3 YEARS)

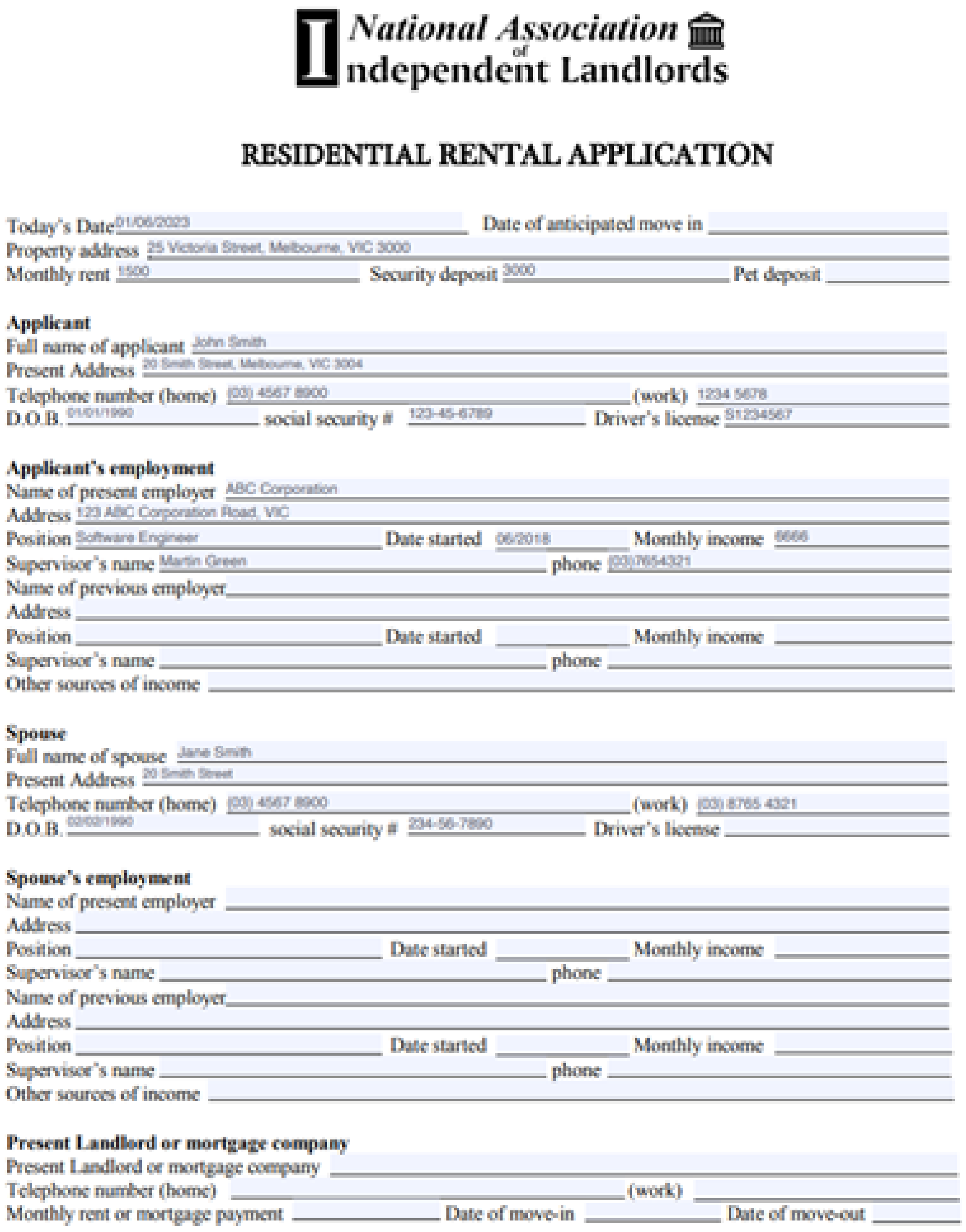
Current Address	20 Smith Street, Melbourne, VIC 3004
Apt#	1A
City	Melbourne
State	VIC
Zip	3004
Month/Year Moved In	06/2020
Reasons for Leaving	End of Lease
Rent $	1400
Owner/Agent	Alice Thompson
Phone	(03) 9876 5432
Previous Address (last 3 years)	10 Jones Lane, Melbourne, VIC 3000
Rent $	1300
Owner/Agent	Bob Johnson
Phone	(03) 8765 4321

PLEASE DESCRIBE YOUR CREDIT HISTORY - Mark with an "X"

	Yes	No
Have you declared bankruptcy in the past seven (7) years?		X
Have you ever been evicted from a rental residence?		X
Have you had two or more late rental payments in the past year?		X
Have you ever willfully or intentionally refused to pay rent when due?		X

- 1 -

Figure A.1: Input Rental Application 1



National Association of Independent Landlords

RESIDENTIAL RENTAL APPLICATION

Today's Date 01/06/2023 Date of anticipated move in
Property address 25 Victoria Street, Melbourne, VIC 3000
Monthly rent 1500 Security deposit 3000 Pet deposit

Applicant
Full name of applicant John Smith
Present Address 20 Smith Street, Melbourne, VIC 3004
Telephone number (home) (03) 4567 8900 (work) 1234 5678
D.O.B. 01/01/1990 social security # 123-45-6789 Driver's license S1234567

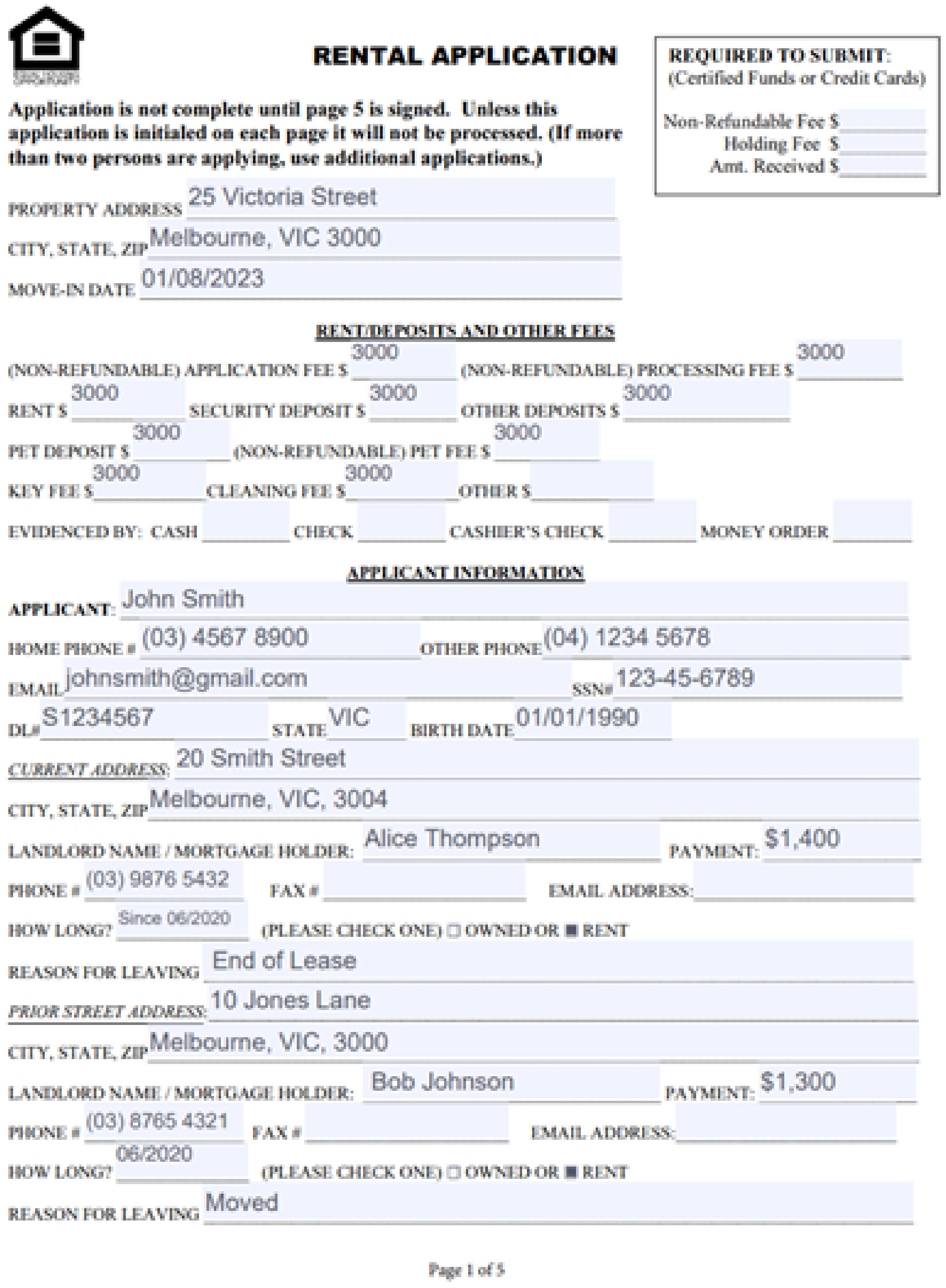
Applicant's employment
Name of present employer ABC Corporation
Address 123 ABC Corporation Road, VIC
Position Software Engineer Date started 06/2018 Monthly income 6666
Supervisor's name Martin Green phone (03)7654321
Name of previous employer
Address
Position Date started Monthly income
Supervisor's name phone
Other sources of income

Spouse
Full name of spouse Jane Smith
Present Address 20 Smith Street
Telephone number (home) (03) 4567 8900 (work) (03) 8765 4321
D.O.B. 02/02/1990 social security # 234-56-7890 Driver's license

Spouse's employment
Name of present employer
Address
Position Date started Monthly income
Supervisor's name phone
Name of previous employer
Address
Position Date started Monthly income
Supervisor's name phone
Other sources of income

Present Landlord or mortgage company
Present Landlord or mortgage company
Telephone number (home) (work)
Monthly rent or mortgage payment Date of move-in Date of move-out

Figure A.2: Input Rental Application 2



RENTAL APPLICATION

REQUIRED TO SUBMIT:
(Certified Funds or Credit Cards)

Non-Refundable Fee $____
Holding Fee $____
Amt. Received $____

Application is not complete until page 5 is signed. Unless this application is initialed on each page it will not be processed. (If more than two persons are applying, use additional applications.)

PROPERTY ADDRESS 25 Victoria Street
CITY, STATE, ZIP Melbourne, VIC 3000
MOVE-IN DATE 01/08/2023

RENT/DEPOSITS AND OTHER FEES

(NON-REFUNDABLE) APPLICATION FEE $ 3000 (NON-REFUNDABLE) PROCESSING FEE $ 3000
RENT $ 3000 SECURITY DEPOSIT $ 3000 OTHER DEPOSITS $ 3000
PET DEPOSIT $ 3000 (NON-REFUNDABLE) PET FEE $ 3000
KEY FEE $ 3000 CLEANING FEE $ 3000 OTHER $____
EVIDENCED BY: CASH ____ CHECK ____ CASHIER'S CHECK ____ MONEY ORDER ____

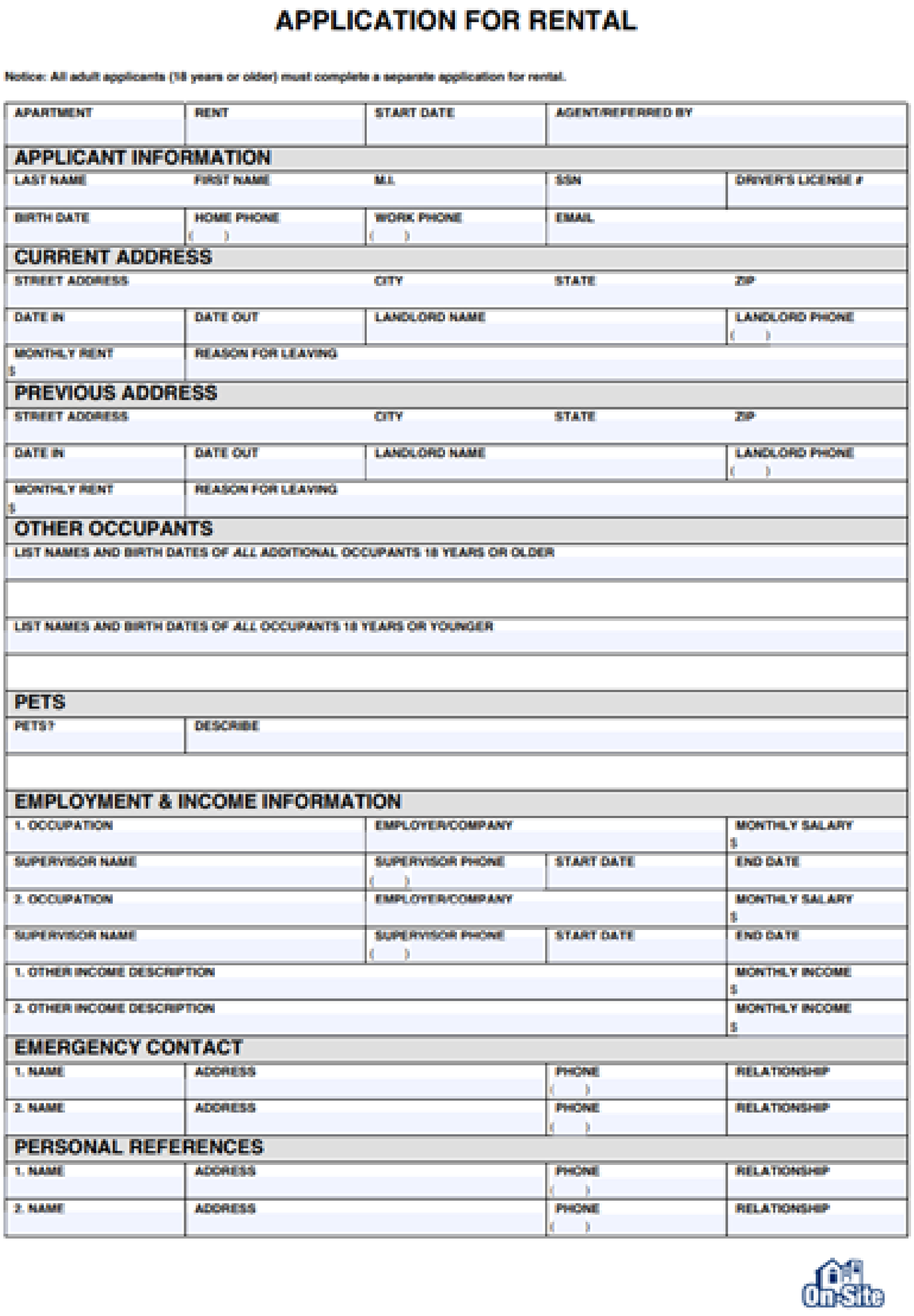
APPLICANT INFORMATION

APPLICANT: John Smith
HOME PHONE # (03) 4567 8900 OTHER PHONE (04) 1234 5678
EMAIL johnsmith@gmail.com SSN# 123-45-6789
DL# S1234567 STATE VIC BIRTH DATE 01/01/1990
CURRENT ADDRESS: 20 Smith Street
CITY, STATE, ZIP Melbourne, VIC, 3004
LANDLORD NAME / MORTGAGE HOLDER: Alice Thompson PAYMENT: $1,400
PHONE # (03) 9876 5432 FAX # ____ EMAIL ADDRESS: ____
HOW LONG? Since 06/2020 (PLEASE CHECK ONE) ☐ OWNED OR ■ RENT
REASON FOR LEAVING End of Lease
PRIOR STREET ADDRESS: 10 Jones Lane
CITY, STATE, ZIP Melbourne, VIC, 3000
LANDLORD NAME / MORTGAGE HOLDER: Bob Johnson PAYMENT: $1,300
PHONE # (03) 8765 4321 FAX # ____ EMAIL ADDRESS: ____
HOW LONG? 06/2020 (PLEASE CHECK ONE) ☐ OWNED OR ■ RENT
REASON FOR LEAVING Moved

Page 1 of 5

Figure A.3: Input Rental Application 3

A.3 Empty Output Form of Rental Application



APPLICATION FOR RENTAL

Notice: All adult applicants (18 years or older) must complete a separate application for rental.

APARTMENT	RENT	START DATE	AGENT/REFERRED BY

APPLICANT INFORMATION

LAST NAME	FIRST NAME	M.I.	SSN	DRIVER'S LICENSE #
BIRTH DATE	HOME PHONE ( )	WORK PHONE ( )	EMAIL	

CURRENT ADDRESS

STREET ADDRESS		CITY	STATE	ZIP
DATE IN	DATE OUT	LANDLORD NAME		LANDLORD PHONE ( )
MONTHLY RENT $	REASON FOR LEAVING			

PREVIOUS ADDRESS

STREET ADDRESS		CITY	STATE	ZIP
DATE IN	DATE OUT	LANDLORD NAME		LANDLORD PHONE ( )
MONTHLY RENT $	REASON FOR LEAVING			

OTHER OCCUPANTS

LIST NAMES AND BIRTH DATES OF *ALL* ADDITIONAL OCCUPANTS 18 YEARS OR OLDER

LIST NAMES AND BIRTH DATES OF *ALL* OCCUPANTS 18 YEARS OR YOUNGER

PETS

PETS?	DESCRIBE

EMPLOYMENT & INCOME INFORMATION

1. OCCUPATION	EMPLOYER/COMPANY		MONTHLY SALARY $
SUPERVISOR NAME	SUPERVISOR PHONE ( )	START DATE	END DATE
2. OCCUPATION	EMPLOYER/COMPANY		MONTHLY SALARY $
SUPERVISOR NAME	SUPERVISOR PHONE ( )	START DATE	END DATE
1. OTHER INCOME DESCRIPTION			MONTHLY INCOME $
2. OTHER INCOME DESCRIPTION			MONTHLY INCOME $

EMERGENCY CONTACT

1. NAME	ADDRESS	PHONE ( )	RELATIONSHIP
2. NAME	ADDRESS	PHONE ( )	RELATIONSHIP

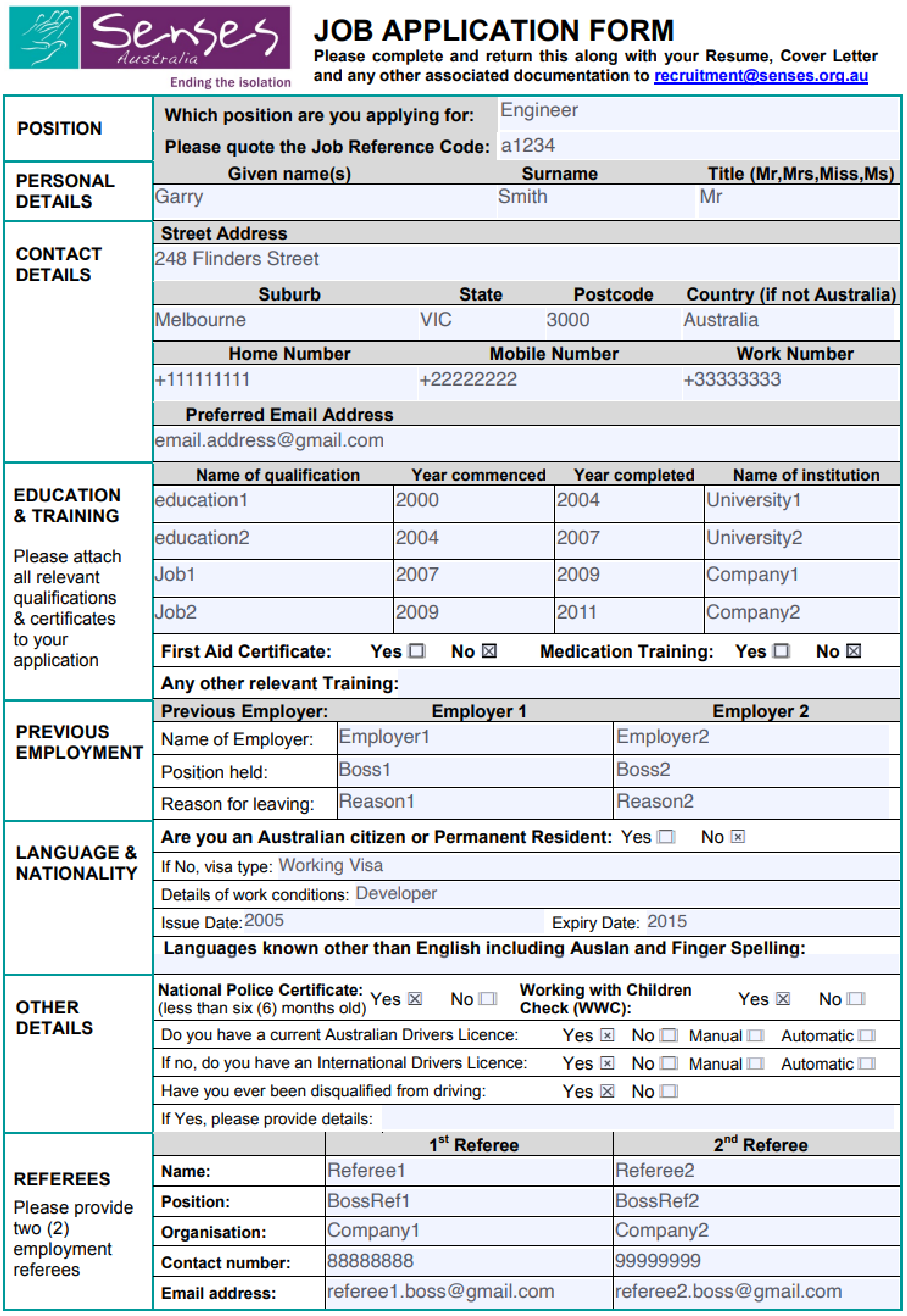
PERSONAL REFERENCES

1. NAME	ADDRESS	PHONE ( )	RELATIONSHIP
2. NAME	ADDRESS	PHONE ( )	RELATIONSHIP

On-Site

Figure A.4: Empty Output Form of Rental Application

A.4 Three Input Job Applications



Senses Australia
Ending the isolation

JOB APPLICATION FORM

Please complete and return this along with your Resume, Cover Letter and any other associated documentation to recruitment@senses.org.au

POSITION	**Which position are you applying for:**	Engineer		
	Please quote the Job Reference Code:	a1234		
PERSONAL DETAILS	**Given name(s)**	**Surname**	**Title (Mr,Mrs,Miss,Ms)**	
	Garry	Smith	Mr	
CONTACT DETAILS	**Street Address**			
	248 Flinders Street			
	Suburb	**State**	**Postcode**	**Country (if not Australia)**
	Melbourne	VIC	3000	Australia
	Home Number	**Mobile Number**	**Work Number**	
	+111111111	+22222222	+33333333	
	Preferred Email Address			
	email.address@gmail.com			

EDUCATION & TRAINING Please attach all relevant qualifications & certificates to your application	**Name of qualification**	**Year commenced**	**Year completed**	**Name of institution**
	education1	2000	2004	University1
	education2	2004	2007	University2
	Job1	2007	2009	Company1
	Job2	2009	2011	Company2
	First Aid Certificate: Yes ☐ No ☒		**Medication Training:** Yes ☐ No ☒	
	Any other relevant Training:			

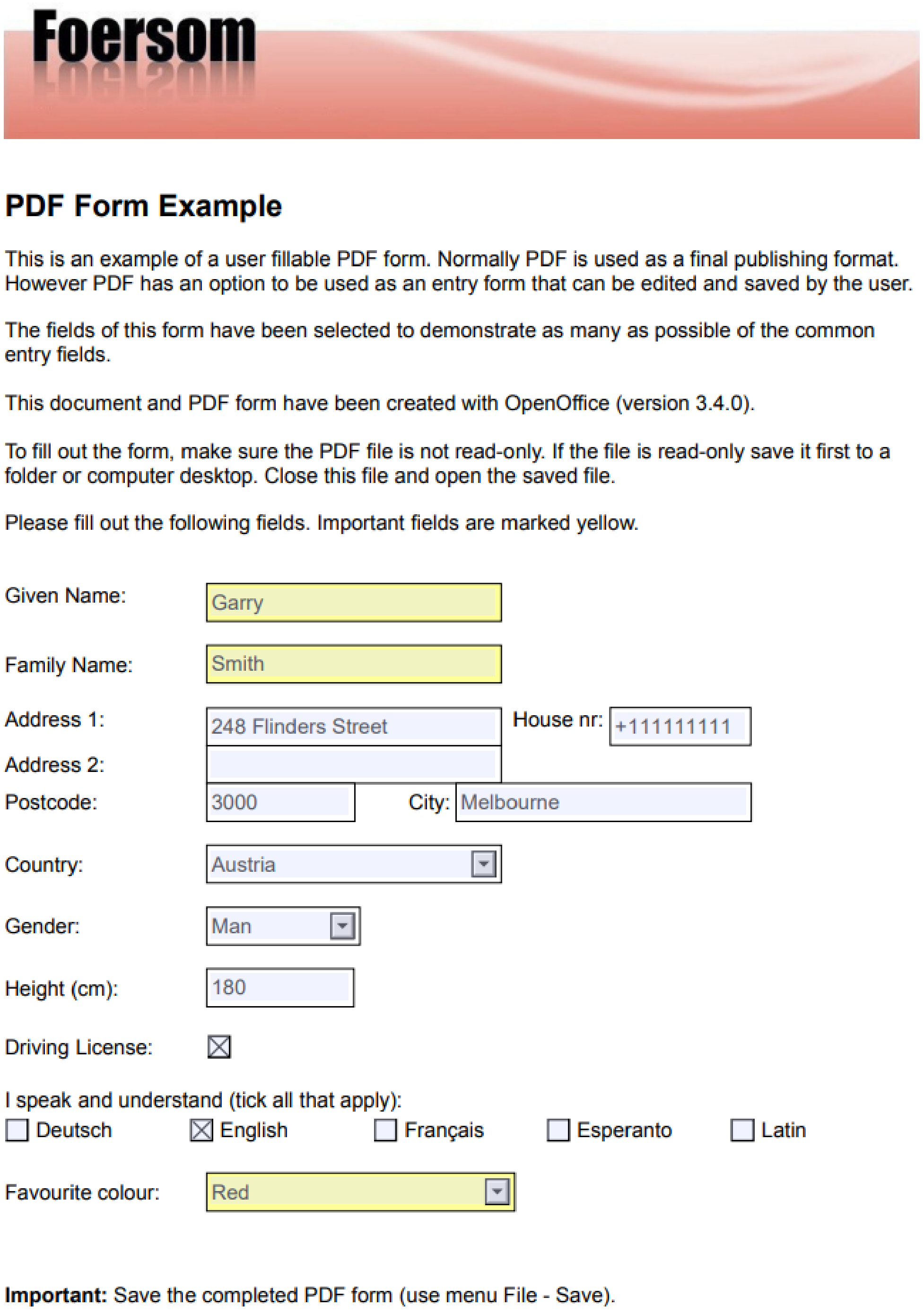
PREVIOUS EMPLOYMENT	**Previous Employer:**	**Employer 1**	**Employer 2**
	Name of Employer:	Employer1	Employer2
	Position held:	Boss1	Boss2
	Reason for leaving:	Reason1	Reason2

LANGUAGE & NATIONALITY	**Are you an Australian citizen or Permanent Resident:** Yes ☐ No ☒	
	If No, visa type: Working Visa	
	Details of work conditions: Developer	
	Issue Date: 2005	Expiry Date: 2015
	Languages known other than English including Auslan and Finger Spelling:	

OTHER DETAILS	**National Police Certificate:** (less than six (6) months old) Yes ☒ No ☐	**Working with Children Check (WWC):** Yes ☒ No ☐
	Do you have a current Australian Drivers Licence:	Yes ☒ No ☐ Manual ☐ Automatic ☐
	If no, do you have an International Drivers Licence:	Yes ☒ No ☐ Manual ☐ Automatic ☐
	Have you ever been disqualified from driving:	Yes ☒ No ☐
	If Yes, please provide details:	

REFEREES Please provide two (2) employment referees		**1st Referee**	**2nd Referee**
	Name:	Referee1	Referee2
	Position:	BossRef1	BossRef2
	Organisation:	Company1	Company2
	Contact number:	88888888	99999999
	Email address:	referee1.boss@gmail.com	referee2.boss@gmail.com

Figure A.5: Input Job Application 1



Foersom

PDF Form Example

This is an example of a user fillable PDF form. Normally PDF is used as a final publishing format. However PDF has an option to be used as an entry form that can be edited and saved by the user.

The fields of this form have been selected to demonstrate as many as possible of the common entry fields.

This document and PDF form have been created with OpenOffice (version 3.4.0).

To fill out the form, make sure the PDF file is not read-only. If the file is read-only save it first to a folder or computer desktop. Close this file and open the saved file.

Please fill out the following fields. Important fields are marked yellow.

Given Name: Garry

Family Name: Smith

Address 1: 248 Flinders Street House nr: +111111111

Address 2:

Postcode: 3000 City: Melbourne

Country: Austria

Gender: Man

Height (cm): 180

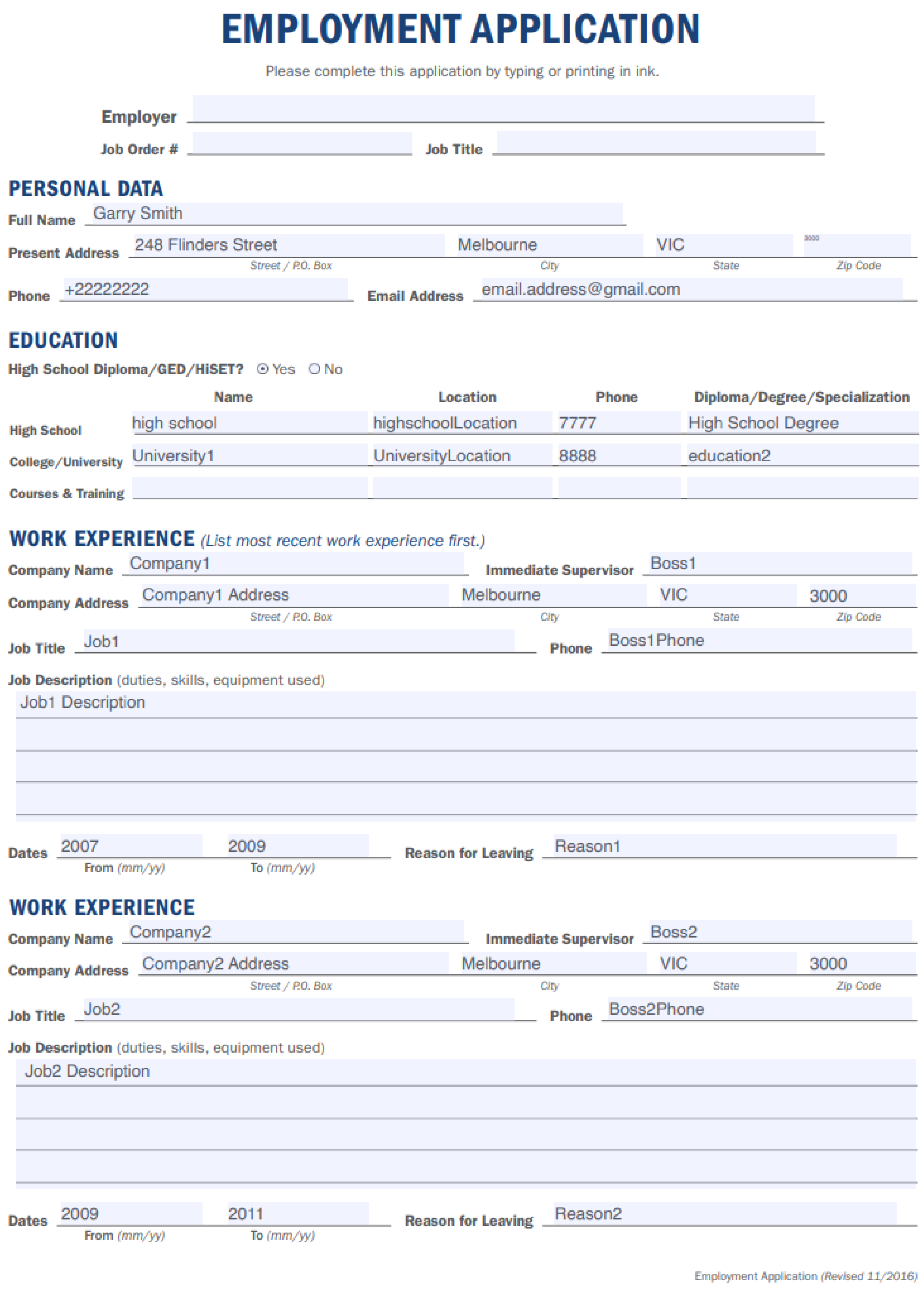
Driving License: ☒

I speak and understand (tick all that apply):
☐ Deutsch ☒ English ☐ Français ☐ Esperanto ☐ Latin

Favourite colour: Red

Important: Save the completed PDF form (use menu File - Save).

Figure A.6: Input Job Application 2



EMPLOYMENT APPLICATION

Please complete this application by typing or printing in ink.

Employer

Job Order # **Job Title**

PERSONAL DATA

Full Name Garry Smith

Present Address 248 Flinders Street (Street / P.O. Box) Melbourne (City) VIC (State) 3000 (Zip Code)

Phone +22222222 **Email Address** email.address@gmail.com

EDUCATION

High School Diploma/GED/HiSET? [✓ Yes] No

	Name	Location	Phone	Diploma/Degree/Specialization
High School	high school	highschoolLocation	7777	High School Degree
College/University	University1	UniversityLocation	8888	education2
Courses & Training				

WORK EXPERIENCE *(List most recent work experience first.)*

Company Name Company1 **Immediate Supervisor** Boss1

Company Address Company1 Address (Street / P.O. Box) Melbourne (City) VIC (State) 3000 (Zip Code)

Job Title Job1 **Phone** Boss1Phone

Job Description (duties, skills, equipment used)

Job1 Description

Dates 2007 (From *(mm/yy)*) 2009 (To *(mm/yy)*) **Reason for Leaving** Reason1

WORK EXPERIENCE

Company Name Company2 **Immediate Supervisor** Boss2

Company Address Company2 Address (Street / P.O. Box) Melbourne (City) VIC (State) 3000 (Zip Code)

Job Title Job2 **Phone** Boss2Phone

Job Description (duties, skills, equipment used)

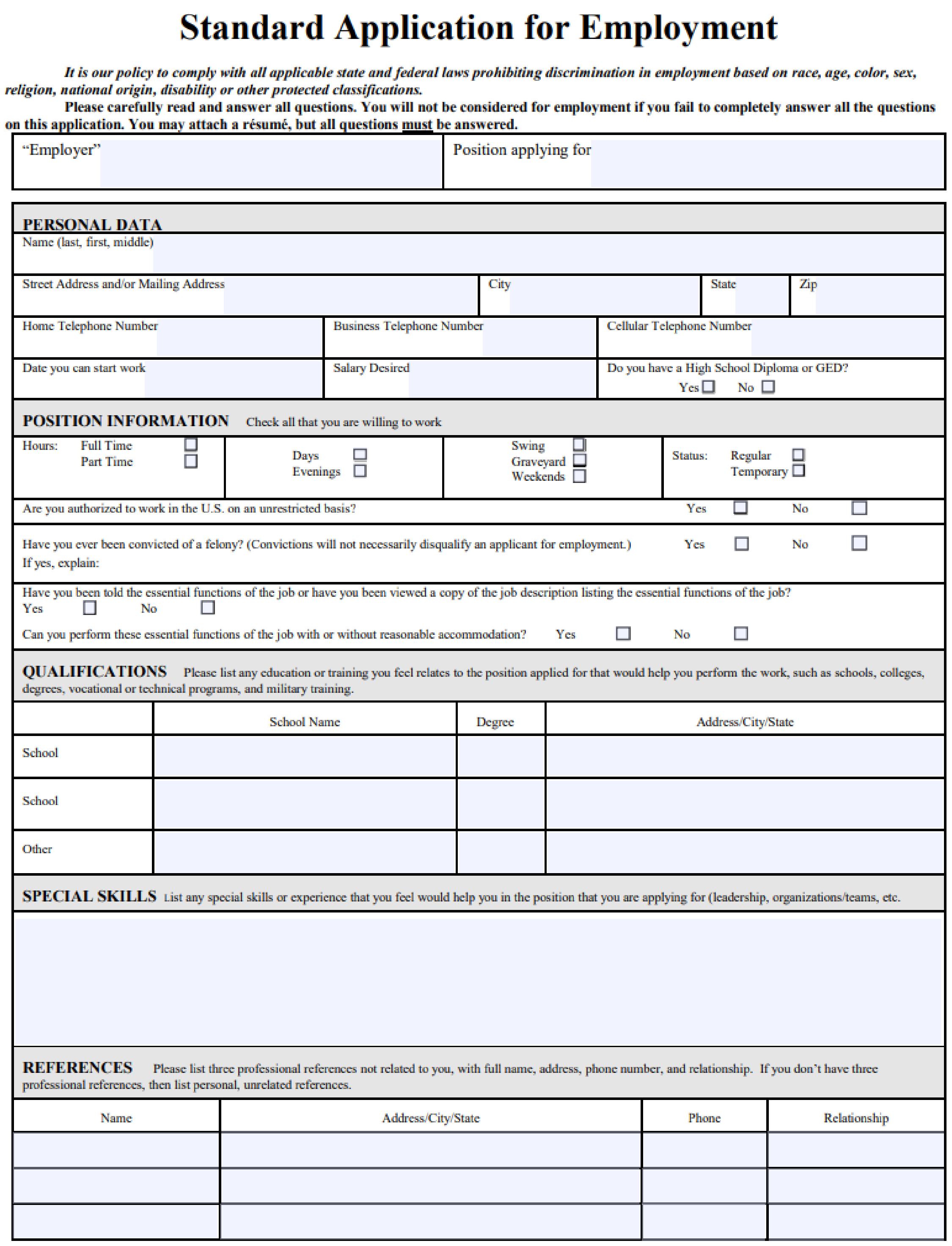
Job2 Description

Dates 2009 (From *(mm/yy)*) 2011 (To *(mm/yy)*) **Reason for Leaving** Reason2

Employment Application *(Revised 11/2016)*

Figure A.7: Input Job Application 3

A.5 Empty Output Form of Job Application



Standard Application for Employment

It is our policy to comply with all applicable state and federal laws prohibiting discrimination in employment based on race, age, color, sex, religion, national origin, disability or other protected classifications.

Please carefully read and answer all questions. You will not be considered for employment if you fail to completely answer all the questions on this application. You may attach a résumé, but all questions must be answered.

"Employer"	Position applying for

PERSONAL DATA

Name (last, first, middle)			
Street Address and/or Mailing Address	City	State	Zip
Home Telephone Number	Business Telephone Number	Cellular Telephone Number	
Date you can start work	Salary Desired	Do you have a High School Diploma or GED? Yes ☐ No ☐	

POSITION INFORMATION Check all that you are willing to work

Hours: Full Time ☐ Part Time ☐	Days ☐ Evenings ☐	Swing ☐ Graveyard ☐ Weekends ☐	Status: Regular ☐ Temporary ☐

Are you authorized to work in the U.S. on an unrestricted basis? Yes ☐ No ☐

Have you ever been convicted of a felony? (Convictions will not necessarily disqualify an applicant for employment.) Yes ☐ No ☐
If yes, explain:

Have you been told the essential functions of the job or have you been viewed a copy of the job description listing the essential functions of the job?
Yes ☐ No ☐

Can you perform these essential functions of the job with or without reasonable accommodation? Yes ☐ No ☐

QUALIFICATIONS Please list any education or training you feel relates to the position applied for that would help you perform the work, such as schools, colleges, degrees, vocational or technical programs, and military training.

	School Name	Degree	Address/City/State
School			
School			
Other			

SPECIAL SKILLS List any special skills or experience that you feel would help you in the position that you are applying for (leadership, organizations/teams, etc.

REFERENCES Please list three professional references not related to you, with full name, address, phone number, and relationship. If you don't have three professional references, then list personal, unrelated references.

Name	Address/City/State	Phone	Relationship

Figure A.8: Empty Output Form of Job Application

Bibliography

Altszyler, E., Sigman, M., and Slezak, D. F. (2017). Corpus specificity in lsa and word2vec: the role of out-of-domain documents.

Bernal, D. G., Giaretta, L., Girdzijauskas, S., and Sahlgren, M. (2021). Federated word2vec: Leveraging federated learning to encourage collaborative representation learning.

Bose, J. (2019). Field label prediction for autofill in web browsers.

Google. (n.d.) (2023). "fill out forms automatically".

Greenleaf, G. (2021). Global data privacy laws 2021: 132 national laws & many bills. privacy laws & business international report.

Heo, D. and Choi, H. (2023). Shared latent space by both languages in non-autoregressive neural machine translation.

Jang, D. and Kim, C.-E. (2023). Exploring the potential of large language models in traditional korean medicine: A foundation model approach to culturally-adapted healthcare. *arXiv preprint arXiv:2303.17807*.

Maeng, K., Colin, A., and Lucia, B. (2019). Alpaca: Intermittent execution without checkpoints.

OpenAI (2023). "function calling with GPT. openai.".

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Poesia, G., Gandhi, K., Zelikman, E., and Goodman, N. D. (2023). Certified reasoning with language models. *arXiv preprint arXiv:2306.04031*.

Qiu, N. and Dai, G. (2012). Design and implement of online intelligent form filling system. In *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, pages 2652–2655. IEEE.

Valentini, F., Rosati, G., Blasi, D., Slezak, D. F., and Altszyler, E. (2021). On the interpretation and significance of bias metrics in texts: a pmi-based approach.

Wang, H. (2021). RankMat: Matrix factorization with calibrated distributed embedding and fairness enhancement. In *2021 the 7th International Conference on Communication and Information Processing (ICCIP)*. ACM.

Wu, S., Luo, X., and Liao, Y. (2020). Personal information classification based on deep learning in automatic form filling system. *International Journal of Computer and Information Engineering*, 14(11):437–445.

Wu, Z., Geiger, A., Potts, C., and Goodman, N. D. (2023). Interpretability at scale: Identifying causal mechanisms in alpaca.

Ye, J., Chen, X., Xu, N., Zu, C., Shao, Z., Liu, S., Cui, Y., Zhou, Z., Gong, C., Shen, Y., Zhou, J., Chen, S., Gui, T., Zhang, Q., and Huang, X. (2023). A comprehensive capability analysis of gpt-3 and gpt-3.5 series models.